2008

# Protocol-level performance analysis and implementation for anti-collision protocols in RFID systems

Mohammed Berhea
*University of Windsor*

# Protocol-Level Performance Analysis and Implementation for Anti-Collision Protocols in RFID Systems

by

**Mohammed Berhea**

A Thesis

Submitted to the Faculty of Graduate Studies through the

Department of Electrical and Computer Engineering in Partial Fulfillment

of the Requirements for the Degree of Master of Applied Science at

The University of Windsor

Windsor, Ontario, Canada

2008

*Your file Votre référence*
*ISBN: 978-0-494-47025-1*
*Our file Notre référence*
*ISBN: 978-0-494-47025-1*

# Declaration of Co-Authorship / Previous Publication

## I. Co-Authorship Declaration

I hereby declare that this thesis incorporates material that is result of joint research, as follows:

In all cases, the primary contributions, derivations, experimental setups, data analysis and interpretation were performed by the author through the supervision of Dr. C. Chen. In addition to supervision, Dr. C. Chen provided the author with the project idea, guidance, and financial help. Dr. C. Chen had also held regular meetings with Dr. Q. M. Jonathan Wu regarding project discussions and follow-ups about the progress and outcome of this project.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include their contributions in my thesis.

I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work.

# II. Declaration of Previous Publication

This thesis includes 2 original papers that have been previously published / submitted for publication in peer reviewed journals, as follows:

| Thesis Chapter | Publication title/full citation | Publication status* |
|---|---|---|
| *Chapter 3, 4, and 5* | M. Berhea, C. Chen, and Q. M. J. Wu, "Protocol-Level Performance Analysis for Anti-Collision Protocols in RFID Systems", In Proceedings of 2008 IEEE International Symposium on Circuits and Systems (ISCAS'08), May 2008, Seattle, USA | *"Accepted for poster session"* |
| *Chapter 3, 4, 5, and 6* | M. Berhea, C. Chen, and Q. M. J. Wu, "Protocol-Level Performance Analysis and Implementation for Anti-Collision Protocols in RFID Systems" | *"Submitted for publication" to IEEE Trans. On Vehicular Technology* |

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Windsor.

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University of Institution.

# Abstract

In this thesis, we provide an analytical approach to evaluate the performance of anti-collision protocols in Radio-frequency identification RFID systems. The analysis is given at protocol-level using performance metrics such as the number of state transitions and clock cycles involved in the protocols' state diagrams, their power and energy consumption. Different protocols were also implemented at different level of abstractions and accurate power analysis was performed in layout level. Discussion and comparison are based on worst-case scenario that represents the process of identifying the last tag when multiple tags are simultaneously available in the system. In particular, a new protocol is proposed for performance improvement. This protocol not only provides fast tag identification mechanism but also reduces tag's power and energy consumptions as compared to other protocols that were covered in this study.

# Acknowledgements

I would like to express my sincere appreciation to my supervisor Dr. Chunhong Chen for his guidance and encouragement. He introduced me to this interesting research area and guided me throughout my thesis with great patience. I would like to thank the members of my thesis committee, Dr. Behnam Shahrrava and Dr. Dan Wu for their advice regarding the research process and their assistance in the preparation of this thesis. Here, I would also like to thank Dr. Rashid Rashidzadeh and Dr. Roberto Muscedere for their incessant help during my chip design.

I am grateful to my wife who accompanied me all these years. Her understanding and support made things easier for me. I am indebted to my parents. They gave me the opportunity to pursue my own life. I would not have reached this milestone in my life without their continuous encouragement and support.

# Table of Contents

**DECLARATION OF CO-AUTHORSHIP / PREVIOUS PUBLICATION ..... III**

I. CO-AUTHORSHIP DECLARATION ..... III

II. DECLARATION OF PREVIOUS PUBLICATION ..... IV

**ABSTRACT ..... VI**

**ACKNOWLEDGEMENTS ..... VII**

**LIST OF FIGURES ..... XII**

**LIST OF TABLES ..... XIV**

**CHAPTER 1 INTRODUCTION ..... 1**

1.1. RFID SYSTEM ..... 1

1.2. OBJECT IDENTIFICATION AND ANTI-COLLISION PROTOCOLS ..... 2

1.3. MOTIVATION AND ACHIEVEMENTS ..... 6

1.4. THESIS ORGANIZATION ..... 7

**CHAPTER 2 REVIEW OF ANTI-COLLISION PROTOCOLS ..... 8**

2.1. COST FUNCTION FOR ANTI-COLLISION PROTOCOLS ..... 8

2.2. BINARY-TREE PROTOCOL ..... 11

2.3. QUERY-TREE PROTOCOL ..... 12

2.4. IMPROVED QUERY-TREE PROTOCOL ..... 14

2.5. COMPARISON OF THE THREE PROTOCOLS ..... 15

**CHAPTER 3 PROTOCOL-LEVEL ANALYSIS OF TREE-BASED ANTI-COLLISION PROTOCOLS ..... 16**

3.1. PROTOCOL-LEVEL ANALYSIS OF BINARY-TREE PROTOCOL AND DETERMINATION OF WORST-CASE SCENARIO ..... 16

*3.1.1. Determination of worst-case scenario ..... 20*

*3.1.2. Derivations of protocol-level performance metrics for Binary-Tree Protocol. ..... 22*

3.2. PROTOCOL-LEVEL ANALYSIS OF QUERY-TREE PROTOCOL ..... 27

3.3. PROTOCOL-LEVEL ANALYSIS OF IMPROVED QUERY-TREE PROTOCOL ..... 33

**CHAPTER 4 A COMBINED BINARY-QUERY-TREE PROTOCOL-THE PROPOSED PROTOCOL ..... 39**

**CHAPTER 5 PROTOCOL-LEVEL COMPARISON OF THE FOUR TREE PROTOCOLS ..... 46**

5.1. PROTOCOL-LEVEL COMPARISON OF THE BINARY, QUERY, AND IMPROVED QUERY TREE PROTOCOLS ..... 46

5.2. PROTOCOL-LEVEL COMPARISON OF THE PROPOSED PROTOCOL WITH BINARY AND IMPROVED QUERY PROTOCOLS ..... 50

**CHAPTER 6 IMPLEMENTATION AND COMPARISON OF THE FOUR PROTOCOLS AT DIFFERENT LEVELS OF ABSTRACTION ..... 52**

6.1. IMPLEMENTATION AND SIMULATION OF THE BINARY-TREE PROTOCOL ..... 53

6.2. IMPLEMENTATION AND SIMULATION OF THE QUERY-TREE PROTOCOL ..... 55

6.3. IMPLEMENTATION AND SIMULATION OF THE IMPROVED QUERY-TREE PROTOCOL ..... 57

6.4. IMPLEMENTATION AND SIMULATION OF THE COMBINED-BINARY-TREE PROTOCOL ..... 59

6.5. LAYOUT LEVEL COMPARISON OF THE FOUR PROTOCOLS ..... 61

*6.5.1. Comparison of the four protocols using "Encounter" ..... 61*

*6.5.2. Comparison of the four protocols in "dfII" ..... 65*

6.5.2.1. Results from Physical Implementation for Binary-Tree Protocol ..... 67
6.5.2.2. Results from Physical Implementation for Query-Tree Protocol ..... 68
6.5.2.3. Results from Physical Implementation for Improved Query-Tree Protocol ..... 71
6.5.2.4. Results from Physical Implementation for Combined-Binary-Tree Protocol ..... 74

**CHAPTER 7 CONCLUSIONS AND FUTURE WORK ..... 77**

7.1. CONCLUSIONS AND CONTRIBUTIONS ..... 77
7.2. FUTURE WORK ..... 78

**REFERENCES ..... 79**

**APPENDIX A RTL CODES AND TEST BENCHES ..... 82**

A.1. RTL CODE AND TEST BENCH FOR THE BINARY-TREE PROTOCOL ..... 83
*A.1.1. RTL Code for the Binary-Tree Protocol ..... 83*
*A.1.2. Test bench for the Binary-Tree Protocol ..... 86*
A.2. RTL CODE AND TEST BENCH FOR THE QUERY-TREE PROTOCOL ..... 88
*A.2.1. RTL Code for the Query-Tree Protocol ..... 88*
*A.2.2. Test bench for the Query-Tree Protocol ..... 91*
A.3. RTL CODE AND TEST BENCH FOR THE IMPROVED QUERY-TREE PROTOCOL ..... 94
*A.3.1. RTL Code for the Improved Query-Tree Protocol ..... 94*
*A.3.2. Test bench for Improved Query-Tree Protocol ..... 97*
A.4. RTL CODE AND TEST BENCH FOR THE QUERY-TREE PROTOCOL ..... 100
*A.4.1. RTL Code for the Proposed Protocol ..... 100*
*A.4.2. Test bench for the Proposed Protocol ..... 104*

**APPENDIX B MATLAB CODES FOR PERFORMANCE COMPARISON ..... 106**

B.1. MATLAB PROGRAM FOR COMPARISON OF BINARY, QUERY, AND IMPROVED QUERY PROTOCOLS ..... 107

B.2. MATLAB PROGRAM FOR COMPARISON OF BINARY, IMPROVED-QUERY, AND COMBINED-BINARY PROTOCOLS ..... 114

**APPENDIX C PERMISSION FROM CO-AUTHORS AND IEEE ..... 121**

C.1. PERMISSION TO USE PREVIOUSLY SUBMITTED/ACCEPTED PAPERS ..... 122

C.2. PERMISSION TO REUSE IEEE-COPYRIGHTED MATERIAL ..... 123

**VITA AUCTORIS ..... 125**

# List of Figures

Figure 1.1: Typical RFID system ..... 1
Figure 1.2: Collision Resolution Protocols ..... 3
Figure 1.3: Example of Binary-Tree algorithm ..... 5
Figure 1.4: Example of Query-Tree algorithm ..... 5
Figure 2.1: Rectification Circuit in a practical tag ..... 9
Figure 2.2: State diagram for Binary-Tree Protocol ..... 11
Figure 2.3: State diagram for Query-Tree Protocol ..... 13
Figure 4.1: State diagram for the Combined-Binary-Query-Tree Protocol. ..... 39
Figure 5.1: Performance comparison of binary-tree, query-tree and improved-query-tree protocols under their worst cases (n = 4) ..... 48
Figure 5.2: Performance comparison of binary-tree, query-tree and improved-query-tree protocols under their worst cases (n = 8) ..... 49
Figure 5.3: Performance comparison of binary-tree, improved query-tree and combined binary-query-tree protocols under their worst cases (n = 4). ..... 51
Figure 6.1: Simulation output for Tag # 15 (N = 4, n = 4): Binary-Tree Protocol. .. 53
Figure 6.2: Schematic for Tag # 15 (N = 4): Binary-Tree Protocol ..... 54
Figure 6.3: Layout for Tag # 15 (N = 4): Binary-Tree Protocol ..... 55
Figure 6.4: Simulation output for Tag# 15 (N = 4, n = 4): Query-Tree Protocol. .... 56
Figure 6.5: Schematic for Tag# 15 (N = 4): Query-Tree Protocol ..... 56
Figure 6.6: Layout for Tag# 15 (N = 4): Query-Tree Protocol ..... 57
Figure 6.7: Simulation output for Tag# 15 (N = 4, n = 4): Improved-Query-Tree Protocol. ..... 58

Figure 6.8: Schematic for Tag# 15 (N = 4): Improved-Query-Tree Protocol ..... 58
Figure 6.9: Layout for Tag# 15 (N = 4): Improved-Query-Tree Protocol..... 59
Figure 6.10: Simulation output for Tag# 15 (N = 4, n = 4): Combined-Query-Tree Protocol. ..... 60
Figure 6.11: Schematic for Tag# 15 (N = 4): Combined -Query-Tree Protocol ..... 60
Figure 6.12: Layout for Tag# 15 (N = 4): Combined -Query-Tree Protocol..... 61
Figure 6.13: Test setup to determine instantaneous power by Binary-Tree Protocol (N = 4, n = 2) ..... 66
Figure 6.14: Results of Simulation for Binary-Tree Protocol (N = 4, n = 2) ..... 67
Figure 6.15: Instantaneous power for Binary-Tree Protocol (N = 4, n = 2) ..... 68
Figure 6.16 Results of Simulation for Query-Tree Protocol (N = 4, n = 2) ..... 70
Figure 6.17: Instantaneous power for Query-Tree Protocol (N = 4, n = 2) ..... 71
Figure 6.18: Results of Simulation for Improved Query-Tree Protocol (N = 4, n = 2) ..... 72
Figure 6.19: Instantaneous power for Improved Query-Tree Protocol (N = 4, n = 2) ..... 73
Figure 6.20: Results of Simulation for Combined-Binary-Query-Tree Protocol (N = 4, n = 2)..... 75
Figure 6.21: Instantaneous power for Combined-Binary-Query-Tree Protocol (N = 4, n = 2)..... 76

# List of Tables

Table 2.1: The inquiry process of Binary-Tree protocol on a 3-tag example ..... 12

Table 2.2: The inquiry process of Query-Tree protocol on a 4-tag example ..... 13

Table 2.3: The inquiry process of improved query-tree protocol on a 4-tag example ..... 14

Table 3.1: Identification Process of Two Tags by Binary-Tree Protocol for N = 4 . 17

Table 3.2: Identification Process of Two Tags (Tag# 10, Tag# 12) by Binary-Tree Protocol for N = 4 ..... 18

Table 3.3: Identification Process of Five Tags by Binary-Tree Protocol for N = 4 . 19

Table 3.4: Distribution of Tags with Binary-Tree Protocol for Worst-Case Scenario ..... 21

Table 3.5: Identification Process of Two Tags by Query-Tree Protocol for N = 5 .. 28

Table 3.6: Commands used for Query-Tree Protocol with corresponding # of transitions and # of clock cycles for N = 4 and N = 5 ..... 29

Table 3.7: Identification Process of Two Tags by Improved Query-Tree Protocol for N = 5 ..... 34

Table 3.8: Identification Process of Four Tags by Improved Query-Tree Protocol for N = 5 ..... 36

Table 4.1: Identification Process of Two Tags by Combined Binary-Query-Tree Protocol for N = 5 ..... 40

Table 4.2: Identification Process of Four Tags by Combined Binary-Query-Tree Protocol for N = 5 ..... 41

Table 5.1: Tabulation of $n_{th}$ for every N that determines the region $n \geq n_{th}$ where the binary-tree has lower power consumption over the two query-tree types ..... 48
Table 6.1: Mapping of Out1 and Out2 to the tag response ..... 52
Table 6.2: Capacitance information for the Binary-Tree Protocol ..... 64
Table 6.3: Implementation results for the Binary, Query, Improved-Query, and Combined-Binary-Query protocols for N = 4 and n = 4 ..... 65

# Chapter 1 Introduction

## 1.1. RFID System

Radio Frequency Identification (RFID) system is an automatic identification system that uses radio frequency to identify objects. The system is consisted of a reader, a tag, and a host computer as shown in Figure 1.1. RFID reader identifies objects through wireless communications with tags which have unique ID and information, and are attached to objects.

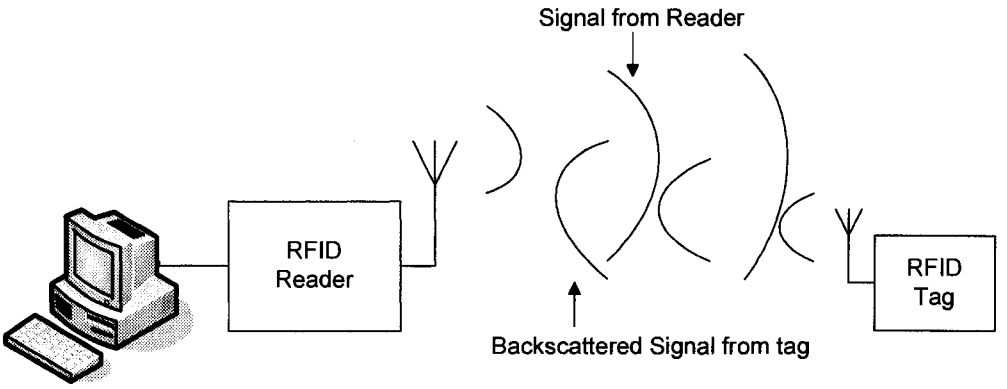


**Figure 1.1: Typical RFID system**

RFID tags (or RFID transponders) can further be divided into three categories [14]. The first is passive tag, which has no power source or battery of its own and relies on the reader as a source of energy for the chip and for communications. The second is semi-passive (battery assisted backscatter) tag which has built in batteries to power-up the chip, but still needs the reader field to communicate back to the reader. The third one is an active tag which has its own battery and for power as well as communication. This thesis is focused on passive RFID tags.

In passive RFID, the reader is designed to transmit energy to, and read information from the tags. The tag is composed of an antenna and a silicon chip that includes basic modulation circuitry and non-volatile memory that permanently stores its ID number [1, 2]. The tag is energized by a time-varying electromagnetic radio frequency (RF) wave that is transmitted by the reader. The RF field generated by a tag reader has three purposes: induce enough power into the tag coil to energize the tag, provide a synchronized clock source to the tag, and to act as a carrier for return data from the tag.

## 1.2. Object Identification and Anti-Collision Protocols

A passive tag is power limited, rather than energy limited. We can reasonably assume that the reader can always keep the energy supply so that a problem such as battery life doesn't exist [1, 13]. Moreover, the available power from the reader field, not only reduces very rapidly with distance, but is also controlled by strict regulations, resulting in a limited communication distance of 4 - 5m when using the UHF frequency band (860 MHz – 930 MHz) [14]. Thus the key focusing point in this thesis is to concentrate on the way of reducing the tag's power consumption, which restricts the tag's maximum possible distance to the reader.

In many existing applications, a single-read RFID tag is sufficient like animal tagging and access control [2]. However, in many new applications such as library books, airline baggage, garment, and retail [14] the simultaneous reading of several tags is necessary. The signals transmitted by the tags may collide as they communicate with a single reader over a shared wireless channel. Thus, anti-collision protocols, which reduce collisions and identify the tags regardless of occurrence of collisions, are required. Therefore, knowing the energy received by the tags is generally very low, optimizing power

consumption through assessing different anti-collision protocols is critical in implementation of the tags which depend on the protocols they use within the system.

In order to resolve collisions, numerous procedures have been developed with the objective of separating the individual participant signals from one another Basically, there are four different procedures [3, 9]: space division multiple access (SDMA), frequency domain multiple access (FDMA), time domain multiple access (TDMA), and code division multiple access (CDMA) as shown in Figure 1.2[9]. SDMA and FDMA are expensive for implementation and are restricted to a few specialized applications while CDMA needs complex circuits and generally rejected for RFID use.

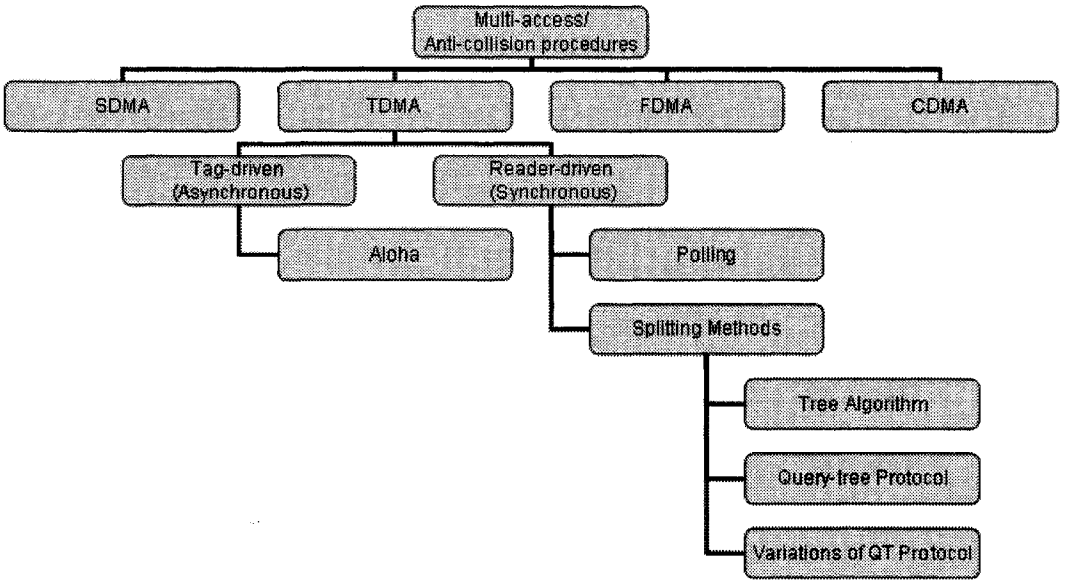


**Figure 1.2: Collision Resolution Protocols**

In RFID systems, TDMA procedures are the popular choice that can broadly be divided into aloha-based (tag-driven) and tree-based protocols (reader-driven) [9, 15]. Aloha-based protocols rely on reducing collisions by separating tag responses by time (or time slots within a frame if using frame-slotted ALOHA) [15]. They are probabilistic in

nature and simple, thus enabling simpler reader and tag implementations. However, they are subject to tag starvation problem, a serious situation where it might either take a long time, or be unable, to identify a tag [7, 8]. In contrast, the tree-based search protocols [4, 5, 15] are deterministic in nature. They are able to read all tags by successively querying nodes at different levels of the tree, with tag IDs distributed on the tree based on their prefix. The guarantee that all tags IDs will be read within a certain time frame has made the tree-based search protocol desirable in many applications. The tree search procedure, however, uses up a lot of reader queries and tag responses by relying on colliding responses of tags to determine which sub-tree to query next. This results in higher energy consumption at the readers and tags (if they are active tags).

In tree-based anti-collision protocols (protocols that thesis is concentrated on), a reader transmits a query (or a feedback) to tags and tags respond to the query by transmitting their ID to the reader. In this process [9], the reader interrogates RFID tags that are in located its vicinity by first asking ''whose serial number's most significant bit (msb) matches with a 0?'' Those RFID tags which do not meet this test then remain silent, and ignore the rest of the interrogation sequence, whilst the rest of them transmit a ''yes that is correct'' answer back to the reader and then await a similar question about the next digit in their binary serial number. The process is repeated until the reader has identified each of the RFID tags in range, following a patter of tree-based search. Figure 1.3 [9] and Figure 1.4 [9] show examples of binary-tree search algorithm, and query-tree search algorithms.

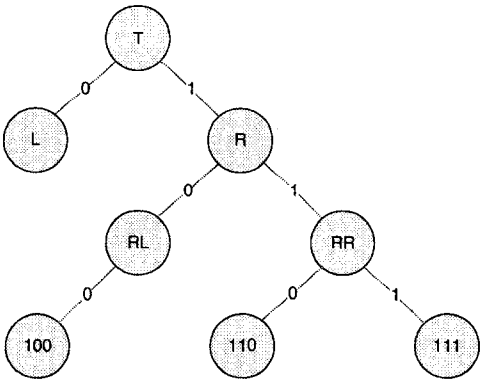


**Figure 1.3: Example of Binary-Tree algorithm**

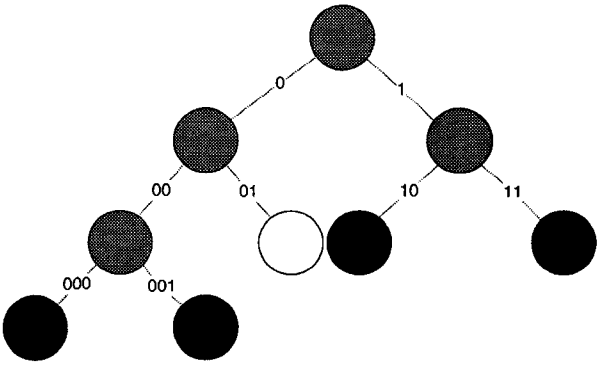


**Figure 1.4: Example of Query-Tree algorithm**

In Figure 1.3, where tags: 100, 110, and 111 are present for interrogation, we see that the reader is sending either a "0" or a "1" at different levels of the tree starting from the "top" towards left (L) or right (R) until it reaches the last leafs to determine the locations of the tags. In Figure 1.4, where tags: 000, 001, 101, and 110 are present, the reader sends different number of prefix (or collection) of bits at different levels during reading process. For example, at level "0" the reader sends a "0" or a "1"; at level "1" it sends two bits: "00" or "01" or "10" or "11"; and so on until it reaches the available tags.

## 1.3. Motivation and Achievements

Previous work (such as [1]) on low power implementation for tree-based protocols was done at circuit-level where a particular protocol could be implemented in different ways with totally different power consumption, leading to an unfair performance evaluation and comparison among various protocols. This work is to assess different performance metrics such as number of transitions, clock cycles, energy, and power consumption of tree-based protocols at protocol-level which is independent of circuit implementation and more efficient.

The basic idea is for a particular N (number of bits that represent and ID) and n (number tags that simultaneous exit for identification) to look at the state diagram for each protocol and to find out the number of state transitions and clock cycles required (and hence its power and energy consumption) before all tags are identified. A general equation for the number of transitions and clock cycles as a function of N and n is then formulated. Then energy consumption which is proportional to the ratio of number of state transitions to number of clock cycles and power consumption which is directly proportional to number of state transitions under equal inquiry time for all protocols were determined. An improved protocol that combines the binary-tree and query-tree is also proposed for further performance improvement.

Each protocol then has been implemented at different levels of abstraction namely, RTL level, Gate level and Layout level. Verilog-XL simulator from Cadence was used for designing the HDL code and for verifying the functionality of the tag. Design_vision from Synopsys was used to synthesize the HDL design into gate-level design. The physical placement and routing of the net-list was performed using Encounter. Finally, the custom physical implementation was done using the Virtuoso Layout Editor.

Accurate power then was estimated at the layout level based on the switching activity and the total capacitance.

## 1.4. Thesis Organization

This thesis is organized as follows.

In Chapter 2 we describe the previous work [1] that deals with circuit-level optimization of power consumption of tags using tree-based protocols. In Chapter 3, for the tree-based protocols mentioned in Chaper2, our analysis will be based on protocol-level instead of circuit-level. For a typical Binary-Tree Protocol case, the number of transitions and clock cycles are tabulated. For this protocol, the condition for the worst case scenario is determined and formula that will allow us to calculate the number of transitions and clock cycles for any ID bits and number of tags is formulated. This chapter also shows similar analysis for Query-Tree and Improved Query-Tree protocols respectively.

Chapter 4 introduces the work of the proposed protocol, Combined Binary-Query-Tree Protocol. In Chapter 5 protocol-level performance evaluation criterion is developed and the three protocols mentioned in Chapter 2 are compared in terms of total number of transitions, total number of clock cycles, energy, and power consumption before a tag is fully read by a reader. Chapter 5 also compares the proposed protocol with binary and improved-query protocols at protocol-level.

Chapter 6 shows the implementation process for the four protocols and compares them at layout-level. In "Encounter", the four protocols are compared in terms of average power while in "dfII", they are compared in terms of instantaneous power. We conclude and describe topics for future work in Chapter 7.

# Chapter 2 Review of Anti-Collision Protocols

This chapter summarizes the work done in [1] that this thesis work is related to.

In the design of RFID tags, the main objective is to reduce power consumption within a tag so as to maximize the working distance between the reader and the tag. Since an anti-collision protocol circuit is the main part of a tag, selecting and implementing a suitable protocol from a low-power consumption point of view, will be a key design criteria. Hence, it is necessary to develop a cost function that can measure the maximum instant power consumption of the circuit [1].

## 2.1. Cost Function for Anti-Collision Protocols

In the design of RFID tags, the main objective is to reduce power consumption within a tag so as to maximize the working distance between the reader and the tag. Since an anti-collision protocol circuit is the main part of a tag, selecting and implementing a suitable protocol from a low-power consumption point of view, will be a key design criteria. Hence, it is necessary to develop a cost function that can measure the maximum instant power consumption of the circuit.

In passive RFID systems, the electromagnetic power a tag can receive from the reader is given by (2.1) [1, 3]:

Where, $P_e = A_e S$ and $A_e = \frac{\lambda^2 G_r}{4\pi}$ $S = \frac{P_{EIRP}}{4\pi r^2}$

There fore,

$$P_e = \frac{\lambda^2 P G_s G_r}{4\pi r^2} \quad (2.1)$$

Where,

S = radiation density, $P_{EIRP}$ = Effective Isotropic Radiated Power, Pe = Power received by tag, P = power injected into reader's antenna, r = distance between reader and tag, Gs = gain of reader's antenna, Gr = gain of tag's antenna, $\lambda$ = wave length of the emitted electro-magnetic wave

Figure 2.1 shows the rectification circuit of a practical tag. When received input power is lower than instantaneous power consumption, the capacitor (C) will provide the required energy compensation. On the other hand, when input power is higher than instantaneous power consumption, transistor (M1) will be turned on to bypass the extra current so as to keep the operating voltage at $V_{dd}$.

$$\frac{1}{2}CV^2{}_{dd} - \frac{1}{2}C(V_{dd} - \Delta V)^2 = E_{t_1,t_2} - P_{in}(t_2 - t_1)T \quad (2.2)$$

Therefore, for any period of time ($t_1$, $t_2$) (measured in clock cycles), conservation of energy will lead us to the following equation:

Where,

$\Delta V$ is the change in capacitance voltage ($V_{dd}$), $P_{in}$ is the input power, $E_{t1,t2}$ is the total energy consumed within the period $t_1$ to $t_2$.

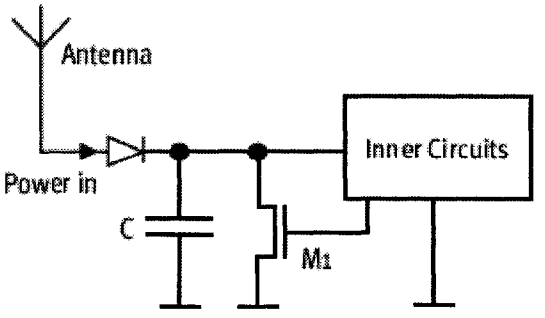


**Figure 2.1: Rectification Circuit in a practical tag**

With the maximum acceptable voltage drop of $V_{drop}$, and assuming $\Delta V << V_{dd}$, $P_{in}$ is simplified to be:

$$P_{in} \geq \frac{V^2{}_{dd} \cdot C_{load}}{(t_2 - t_1)T} \left( \sum_{t_1}^{t_2} A_i - \frac{C \cdot V_{drop}}{V_{dd} \cdot C_{load}} \right) \tag{2.3}$$

Where $A_i$ is number of 0 to 1 transitions within clock cycle i and $C_{load}$ is the load capacitance of the circuit.

Hence, for any arbitrary anti-collision protocol,

$$P_{in,\min} = Max\{P(t_2, t_1) | t_2, t_1 \varepsilon T_{inq}\} \tag{2.4}$$

Where P ($t_1$, $t_2$) represents the right-hand equation of (2.3) and $T_{inq}$ is the total inquiry time for a tag to be totally read by the reader. $P_{in,min}$, the lower bound for input power of tags, is defined as the cost function used to evaluate different protocols.

For fair power consumption comparison among different protocols, equal inquiry time ($T_{INQ}$) for all protocols need to be considered. Hence, with $T_{INQ} = T \cdot Cyc$, where Cyc is total number of clock cycles needed to fully read a tag, $P(t_1, t_2)$ will be adjusted to the following equation:

$$\begin{aligned} P(t_2, t_1) &= \frac{Cyc \cdot V^2{}_{dd} \cdot C_{load}}{(t_2 - t_1)T_{INQ}} \left( \sum_{i=t_1}^{t_2} A_i - \frac{C \cdot V_{drop}}{V_{dd} \cdot C_{load}} \right) \\ &\propto \frac{Cyc}{(t_2 - t_1)} \left( \sum_{i=t_1}^{t_2} A_i - \frac{C \cdot V_{drop}}{V_{dd} \cdot C_{load}} \right) \end{aligned} \tag{2.5}$$

Equation (2.5) will be used as a cost function and with C or $V_{drop}$, it measures the maximum instant power consumption of the circuit. Based on this cost function, different protocols will be evaluated in the next sections.

## 2.2. Binary-Tree Protocol

Binary-Tree Protocol enables a reader to successfully read all tags within its reach by using binary search algorithm [1, 9]. The tags are assumed to have an ID-bit module, and the reader sends one bit at a time during the enquiry. The tags with the same received bit progressively increment their bit position to send the next bit in their ID. If the tags receive a different inquiring bit, they will go to the state of "standby" and stay there until a tag is identified (or killed) with the rest of the tags being reset. During an enquiry, the reader sends '0' when it senses a collision. Otherwise, it uses the received bit from the tag as the next inquiring bit.

The state diagram used by the tag for this protocol is shown in Figure 2.2 [1]

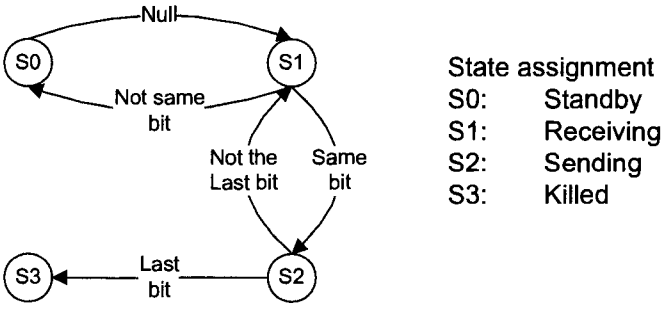


**Figure 2.2: State diagram for Binary-Tree Protocol**

An example shown in Table 2.1 [1] details what steps a reader need to use to fully identify three tags if the tags are using Binary-Tree Protocol. In the table, under "Tags answer" column, * stands for collision situation and empty space represents no-response situation.

After implementing this protocol and analyzing its cost function, the following formula is derived [1]:

$$COST_{binary} = [2k(n-1)+3]\left(10 - \frac{1}{2n-1}R\right) \quad (2.6)$$

$$Where, R = \frac{C \cdot V_{drop}}{C_{load} \cdot V_{dd}} \quad (2.7)$$

Here "n" represents the length of the ID, and "k" represents the distribution of tags (total number of tags simultaneously available for interrogation.

**Table 2.1: The inquiry process of Binary-Tree protocol on a 3-tag example**

| Reader sends | Tags answer | Identified tag (remaining tags reset) |
|---|---|---|
| 0 | * | |
| 0 | 1 | 001 |
| 0 | 1 | |
| 1 | 1 | 011 |
| 0 | | |
| 1 | 0 | |
| 0 | 0 | 100 |

## 2.3. Query-Tree Protocol

Unlike the binary-tree protocol, query-tree protocol is a memory-less one where the tags do not have to remember their inquiring history [1, 4, 9]. In this protocol, the reader first sends a "start" command ("Null" followed by another "Null") to all the tags residing in its vicinity, and then reads the responses from the tags to determine if there is a collision. If no collisions occur, the reader will identify a particular tag. If a collision is detected, the reader will send a prefix that contains the bits sent by the tags prior to the collision bit, followed by a masking bit of a '0' at one time and a '1' at the next enquiry [1]. The tags whose first bits match the sent prefix will then send their remaining bits. This prefix-forming and sending process continues until all the tags are successfully identified.

The state diagram used by the tag for this protocol is shown in Figure 2.3 [1]

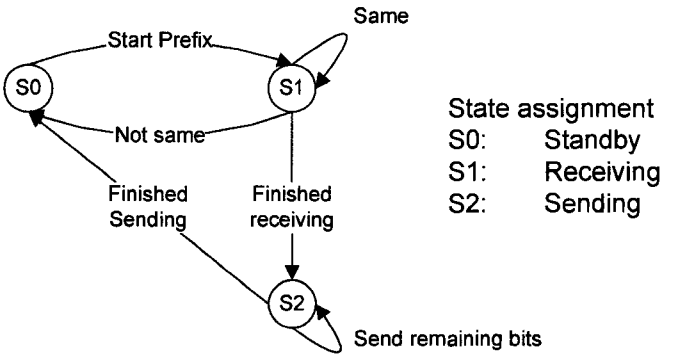


**Figure 2.3: State diagram for Query-Tree Protocol**

An example elaborating the process of identification of four tags (with 4-bit ID) that uses Query-Tree protocol is shown in Table 2.2 [1].

**Table 2.2: The inquiry process of Query-Tree protocol on a 4-tag example**

| Reader sends | Tags answer | Identified tag (remaining tags reset) |
|---|---|---|
| start | **** | |
| 0 | 0*1 | |
| 000 | 1 | 0001 |
| 001 | 1 | 0011 |
| 1 | *00 | |
| 10 | 00 | 1000 |
| 11 | 00 | 1100 |

Also, after implementing this protocol and analyzing its cost function, the following formula is derived [1]:

$$COST_{query} = [k(2n+8)-(n+3)]\left(8-\frac{m^2-9m+4}{2(n+4)(m+1)+4}-\frac{1}{(n+4)(m+1)+2}R\right) \quad (2.8)$$

$$Where, m = \lceil \log_2 k \rceil \quad (2.9)$$

Where R is given by (2.7) and m is the depth of the tree.

## 2.4. Improved Query-Tree Protocol

This protocol is similar to the query-tree protocol, the difference lies on the action that will be taken by the reader when it senses collision. In case of query-tree, tags will continue to send the remaining of their ID bits, even though there is a collision. On the other hand, in Improved Query-Tree-Protocol, when the reader senses a collision, it issues a 'Stop Send" command ("SS") to all the tags to halt transmission. This process will then reduce unnecessary "sending" operations by the tags and effectively reducing power consumption.

The state diagram for this protocol is similar to Figure 2.3 except that the "Finished Sending" edge should be replaced by "Stop Sending" edge.

The Improved query-tree search process in identification of the same four tags that were used in Table 2.2 is shown in Table 2.3 [1]:

**Table 2.3: The inquiry process of improved query-tree protocol on a 4-tag example**

| Reader sends | Tags answer | Identified tag (remaining tags reset) |
|---|---|---|
| start | * | |
| 0 | 0* | |
| 000 | 1 | 0001 |
| 001 | 1 | 0011 |
| 1 | * | |
| 10 | 00 | 1000 |
| 11 | 00 | 1100 |

After implementing this protocol, the following formula is derived for its cost function [1]:

$$COST_{imprved} = [k(n+m+8)-3]\left(8-\frac{m^2-33m-26-2R}{2n+m^2+11m+12}\right) \qquad (2.10)$$

Where R is given by (2.7) and m is given by (2.9).

## 2.5. Comparison of the three protocols

After drawing curves using the cost functions developed in equations (2.6), (2.8), and (2.10), the following general information is deduced [1]:

- With increase in n, the probability of the Query-Tree protocol outperforming the Binary-Tree protocol in terms of power consumption is better. Especially, with small values of R, the performance of Query-Tree protocol as compared to the Binary-Tree protocol is significant.

- Since Improved Query-Tree protocol is always better than Query-Tree protocol, whatever observation made between the Query-Tree protocol and the Binary-Tree protocol holds true when comparing Improved Query-Tree protocol with Binary-tree protocol with added performance in case of Improved query-tree protocol.

By omitting the low power items [13] shows that:

For $R \leq 14.4n - 274.2$, $COST_{improved} \leq COST_{binary}$

For $R \leq 4n - 75$, $COST_{query} \leq COST_{binary}$

For $R \leq 153.96$, $COST_{improved} \leq COST_{query}$

# Chapter 3 Protocol-Level Analysis of Tree-based Anti-Collision Protocols

In this chapter analysis of the three tree-based protocols that were discussed in Chapter 2 will be performed at the protocol level. As mentioned in Chapter 1 and shown in Chapter 2, the previous work was concentrated on the determination of instant power consumption of circuits for comparison of different tree-based anti-collision protocols. But a particular protocol could be implemented in different ways bringing about different power consumptions hence; performance evaluation and comparison among various protocols need to be done at protocol level. Therefore, in this work protocol-level performance metrics such as number of transitions, clock cycles, energy, and power consumption will be used to compare different tree-based protocols.

## 3.1. Protocol-Level Analysis of Binary-Tree Protocol and determination of worst-case scenario

In what follows, the performance of this protocol is studied based on the worst-case scenario in terms of total number of state transitions and clock cycles required before it is identified. As will be seen, the results depend on the number of bits (denoted by N) for the tags, the ID of a particular tag and the ID distribution of other tags present for interrogation (denoted by n).

In general, the tag identification process can be shown in details based on Figure 2.2 (State diagram for Binary-Tree Protocol). Table 3.1 shows an example for N = 4 with 2 tags present. In this table, the 1st column represents the command sent by the reader, the 2nd column is the present state (PS) of the tag, and the following five columns show the number of transitions, $T_{ij}$, that the tag makes from state $S_i$ to state $S_j$. The Clk column in the table denotes the number of clock cycles required, while the NS column indicates the next state. The status of tags that are interacting with the reader is shown in the last tag column, where the two tags #12 (representing ID#1100) and #14 (representing ID#1110) are shown to be read at the 10th and 17th clock cycle, respectively.

**Table 3.1: Identification Process of Two Tags by Binary-Tree Protocol for N = 4**

| Reader | PS | $T_{0-1}$ | $T_{1-0}$ | $T_{1-2}$ | $T_{2-1}$ | $T_{2-3}$ | Clk | NS | Tag |
|---|---|---|---|---|---|---|---|---|---|
| Null | S0 | 1 | | | | | 1 | S1 | |
| 0 | S1 | | 1 | | | | 1 | S0 | |
| | S0 | | | | | | 1 | S0 | No Response |
| Reset | S0 | 1 | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | 1 | S2 | |
| | S2 | | | | 1 | | 1 | S1 | |
| 1 | S1 | | | 1 | | | 1 | S2 | |
| | S2 | | | | 1 | | 1 | S1 | |
| 0 | S1 | | 1 | | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | 1 | **S3**, S0 | #12 (1100) |
| Reset | S0 | 1 | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | 1 | S2 | |
| | S2 | | | | 1 | | 1 | S1 | |
| 1 | S1 | | | 1 | | | 1 | S2 | |
| | S2 | | | | 1 | | 1 | S1 | |
| 1 | S1 | | | 1 | | | 1 | S2 | |
| | S2 | | | | | 1 | 1 | S3 | #14 (1110) |
| Total | - | 3 | 2 | 5 | 4 | 1 | 17 | - | - |

During the reading process, both tags will hold the same state until they reach the 9th clock cycle. At the 9th clock cycle, when the reader sends a masking bit of "0" (note that in the 8th clock cycle, the reader receives a conflict because Tag# 12 sends a "0" and Tag# 14 sends a "1"), Tag# 12 jumps to S2 and Tag# 14 jumps to S0. Hence, in the NS column, at the 9th and 10th clock cycles, and in the PS column, at 10th clock cycle, two

states are written to indicate the states of Tag# 12 and Tag# 14, where the state with the bold font representing the state of Tag# 12. Once Tag #12 is identified (after the $10^{th}$ clock cycle), the state transitions shown in the table are made by Tag# 14. It takes 15 numbers of transitions for Tag# 14 to be identified.

**Table 3.2: Identification Process of Two Tags (Tag# 10, Tag# 12) by Binary-Tree Protocol for N = 4**

| Reader | PS | $T_{0-1}$ | $T_{1-0}$ | $T_{1-2}$ | $T_{2-1}$ | $T_{2-3}$ | Clk | NS | Tag |
|---|---|---|---|---|---|---|---|---|---|
| Null | S0 | 1 | | | | | 1 | S1 | |
| 0 | S1 | | 1 | | | | 1 | S0 | |
| | S0 | | | | | | 1 | S0 | No Response |
| Reset | S0 | 1 | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | 1 | S2 | |
| | S2 | | | | 1 | | 1 | S1 | |
| 0 | S1 | | | 1 | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | 1 | **S1**, S0 | |
| 1 | **S1**, S0 | | | | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | 1 | **S3**, S0 | #10 (1010) |
| Reset | S0 | 1 | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | 1 | S2 | |
| | S2 | | | | 1 | | 1 | S1 | |
| 1 | S1 | | | 1 | | | 1 | S2 | |
| | S2 | | | | 1 | | 1 | S1 | |
| 0 | S1 | | | 1 | | | 1 | S2 | |
| | S2 | | | | | 1 | 1 | S3 | #12 (1100) |
| Total | - | 3 | 1 | 5 | 3 | 1 | 17 | - | - |

For comparison, we draw a table, Table 3.2 that shows the reading process if the two tags are Tag# 10 and Tag# 12. From Table 3.2, the total number of transitions that Tag# 12 makes before it is read is 13, which is lower than 15 compared to the result obtained from Table 3.1. In fact, if we consider any combinations of two tags, the total number of transitions that we can obtain will always be lower than the combination shown in Table 3.1. Hence, for any two tags to be identified with this protocol, the highest possible number of state transitions occurs when the ID of one tag is 1110 (Tag# 14) or 1111 (Tag# 15) and the ID of the other is 1100 (Tag# 12) or 1101(Tag# 13). This is called the **worst-case scenario**. Note that if Tag# 14 (1110) and Tag# 15 (1111) simultaneously appear for interrogation, the reader could be able to identify any one of them by just

reading the last bit. If the last bit is 0, then it is Tag# 14; if the last bit is 1, then it is Tag# 15; if the expected last bit turned out to be a collision, then both Tag# 14 and Tag# 15 are present.

**Table 3.3: Identification Process of Five Tags by Binary-Tree Protocol for N = 4**

| Reader | PS | $T_{0-1}$ | $T_{1-0}$ | $T_{1-2}$ | $T_{2-1}$ | $T_{2-3}$ | Clk | NS | Tag |
|---|---|---|---|---|---|---|---|---|---|
| Null | S0 | 1 | | | | | 1 | S1 | |
| 0 | S1 | | 1 | 0 | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | 1 | **S1**, S0 | |
| 1 | **S1**, S0 | | | | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | 1 | **S1**, S0 | |
| 1 | **S1**, S0 | | | | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | 1 | **S3**, S0 | #6 |
| Reset | S0 | 1 | | | | | 1 | S1 | |
| 1 | S1 | | 0 | 1 | | | 1 | S2 | |
| | S2 | | | | 1 | 0 | 1 | S1 | |
| 0 | S1 | | 1 | 0 | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | 1 | **S1**, S0 | |
| 0 | **S1**, S0 | | | | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | 1 | **S3**, S0 | #8 |
| Reset | S0 | 1 | | | | | 1 | S1 | |
| 1 | S1 | | 0 | 1 | | | 1 | S2 | |
| | S2 | | | | 1 | 0 | 1 | S1 | |
| 0 | S1 | | 1 | 0 | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | 1 | **S1**, S0 | |
| 1 | **S1**, S0 | | | | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | 1 | **S3**, S0 | #10 |
| Reset | S0 | 1 | | | | | 1 | S1 | |
| 1 | S1 | | 0 | 1 | | | 1 | S2 | |
| | S2 | | | | 1 | 0 | 1 | S1 | |
| 1 | S1 | | 0 | 1 | | | 1 | S2 | |
| | S2 | | | | 1 | 0 | 1 | S1 | |
| 0 | S1 | | 1 | 0 | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | 1 | **S3**, S0 | #12 |
| Reset | S0 | 1 | | | | | 1 | S1 | |
| 1 | S1 | | 0 | 1 | | | 1 | S2 | |
| | S2 | | | | 1 | 0 | 1 | S1 | |
| 1 | S1 | | 0 | 1 | | | 1 | S2 | |
| | S2 | | | | 1 | 0 | 1 | S1 | |
| 1 | S1 | | 0 | 1 | | | 1 | S2 | |
| | S2 | | | | 0 | 1 | 1 | S3 | #14 |
| Total | - | 5 | 4 | 7 | 6 | 1 | 35 | - | - |

### 3.1.1. Determination of worst-case scenario

In order to find out the worst-case scenario for any value of N with a given number of tags, we analyze the number of state transitions and clock cycles required for different tags $i^{th}$ N bits in their ID, and summarize the results in Table 3.4 which can be used to determine the ID distribution of the tags that yield a worst-case scenario. In this table, the tags whose most significant bits match in a particular pattern are listed in a group. For example, for a tag represented with a 4-bit ID, Tag# 14 (ID of 1110) and Tag# 15 (ID of 1111) belong to the same group (group # 4) or Tag# 8 (1000), Tag# 9 (1001), Tag# 10 (1010), and Tag# 11 (1011), belong to the same group (group # 2) as shown in Table 3.4.

From Table 3.1 we can see that after Tag# 12 is read, Tag# 14 (or Tag# 15) takes 7 numbers of transitions before it is read (that is, by counting the number of transitions it makes from S0 to S3). This same tag (Tag# 14) took 6 number of transitions while Tag# 12 is being read (that is from the $4^{th}$ clock cycle up to the $10^{th}$ clock cycle in Table 3.1). If we see Table 3.3, Tag# 14 will take 4 numbers of transitions while Tag# 10 is being read. Based on similar observations, Table 3.4 is constructed to group distribution of tags for any N. Hence, any combination of the tags (except for those that belong to same group, differ only in their least significant bit, and can thus be identified simultaneously with the number of transitions for this particular group) will each have the number of transitions with the group.

For instance, if N = 5, the worst-case scenario for simultaneous existence of four tags for reading, requires their distribution to be as follows: one tag from group 5 (Tag# 30 or #31), one tag from group 4 (Tag# 28 or #29), and two tags from group 3 (say, tags #24 and #26), where the worst-case tag being Tag# 30 or Tag# 31. In group 3, simultaneous existence of tags #24 and #25 will not yield a worst-case scenario as they are in the same

group but differ only in their least significant bit and thus can be identified with a total of 6 transitions.

The grouping of tags that determines the worst-case scenario will remain the same for the rest of the tree protocols that we use in this work. Hence, Table 3.4 will stay the same for the query-tree, improved query-tree, and the combined-binary-tree protocols, except for the # of transitions and # of clock cycles, which will be different for different protocols.

**Table 3.4: Distribution of Tags with Binary-Tree Protocol for Worst-Case Scenario**

| N = 4 | | | |
|---|---|---|---|
| group # | pattern of ID bits (tag #s) with * = 0 or 1 | # of transitions | # of clock cycles |
| 1 | 0*** (0~7) | 2 | 7 |
| 2 | 10**(8, 9, 10,11) | 4 | 7 |
| 3 | 110*(12,13) | 6 | 7 |
| 4 | 111*(14,15) | 7 | 7 |
| N = 5 | | | |
| 1 | 0**** (0~15) | 2 | 9 |
| 2 | 10***(16~23) | 4 | 9 |
| 3 | 110**(24~27) | 6 | 9 |
| 4 | 1110*(28,29) | 8 | 9 |
| 5 | 1111*(30,31) | 9 | 9 |
| For any value of N | | | |
| 1 | 0**...** (0~($2^N$/2)–1) | 2 | 2N –1 |
| 2 | 10** ...*($2^N$/2~ ($2^N$/2+$2^N$/4)–1) | 4 | 2N –1 |
| . | . | . | . |
| . | . | . | . |
| . | . | . | . |
| N – 1 | 11...10*($2^N$–4, $2^N$–3) | 2N –2 | 2N –1 |
| N | 11...11*($2^N$–2, $2^N$–1) | 2N –1 | 2N –1 |

### 3.1.2. Derivations of protocol-level performance metrics for Binary-Tree Protocol

Let n, $T_B$ and $CLK_B$ are the number of tags available, the total number of transitions and the number of clock cycles required under the worst-case, respectively. For this protocol and for the rest of the other protocols discussed in this work, for the worst-case tag, formulas to calculate $T_B$ and $CLK_B$ in terms of N and n are derived. The formulas are derived for $2 \leq n \leq 2^{N-1}$ (for $2^{N-1} \leq n \leq 2^N$, the number of transitions and clock cycles will be the same as for $n = 2^{N-1}$ because any two tags that belong to same group but differ only in their least significant bit can be identified at the same time).

For this protocol:

$$T_B = T_{0-1} + T_{1-0} + T_{1-2} + T_{2-1} + T_{2-3} \qquad (3.1)$$

Let us first determine $CLK_B$ (n, N) and $T_B$ (n, N) for the worst-case tag (Tag# 14 or Tag# 15), for N = 4 and for different values of n:

From Table 3.1, where N = 4 and n = 2,

$CLK_B$ (2, 4) = 3 + 7 + 7;

Where,

- The first term (= 3) is the number of clock cycles needed during the "No Response" read cycle.
- The second term (= 7) and the last term (= 7) is number of clock cycles needed to read Tag# 12 and Tag# 14 respectively.

$T_B$ (2, 4) = 2 + 6 + 7 = 15;

Where,

- The first term (= 2) is the sum of number of transitions during the "No Response" read cycle.
- The second term (= 6) is the sum of number of transitions during the reading of Tag# 12
- The last term (= 7) is the sum of number of transitions during the reading of Tag# 14.

From Table 3.3, where N = 4 and n = 5,

$CLK_B$ (5, 4) = 7 + 7 + 7 + 7 + 7 = 35

Where, 7 in $CLK_B$ (5, 4) shows the number of clock cycles that Tag# 14 makes during the reading of a single tag and 7 is repeated 5 times since there are 5 tags.

$T_B$ (5, 4) = 2 + 4 + 4 + 6 + 7 = 23;

Where,

- The first term (= 2) is the sum of number of transitions during the reading of Tag# 6 (0110). Since "msb" of Tag# 6 is "0", when the reader reads Tag# 6, the worst-case tag (Tag# 14) will jump from S0 to S1 and back to S0 for a total of 2 state transitions. Similarly, when any tag with "msb" = 0 is read, the worst-case tag will take 2 transitions.

- The rest of the terms: 4, 4, 6, and 7 comes from total state transitions that Tag# 14 makes during the reading of Tag# 8, Tag# 10, Tag# 12, and Tag# 14 respectively.

Similarly, if we were to construct a table for N = 4 and n = 8, consisting of Tag# 0, 2, 4, 6, 8, 10, 12 and 14 for a worst-case tag distribution, then the number of clock cycles and number of transitions for Tag# 14 will be:

$$CLK_B (8, 4) = 8 * 7 = 56$$

And

$$T_B (8, 4) = 2 + 2 + 2 + 2 + 4 + 4 + 6 + 7 = 29;$$

Where, the four 2s in $T_B$ (8, 4) come from reading Tag# 0, 2, 4 and, 6

Note that for n > 8, then two tags with the same group (shown as Table 3.4) need to exit simultaneously for interrogation. For example, for n = 9, Tag# 0, 1, 2, 4, 6, 8, 10, 12 and 14. Since Tag# 0 and Tag# 1 belong to the same group and can be read simultaneously, $CLK_B (9, 4) = CLK_B (8, 4)$ and $T_B (9, 4) = T_B (8, 4)$.

Hence for N = 4, if n > 8, then

$$CLK_B (n, 4) = CLK_B (8, 4) \text{ and } T_B (n, 4) = T_B (8, 4)$$

For N = 5, through constructing and analyzing similar table as Table 3.1 or 3.3, the number of clock cycles and state transitions are calculated as shown:

Number of clock cycles (N = 5):

n = 2: $CLK_B = 3 + 2*9 = 21$

n = 8: $CLK_B = 3 + 8*9 = 75$

n = 9: $CLK_B = 9*9 = 81$

n = 10: $CLK_B = 10*9 = 90$

Number of state transitions (N = 5):

n = 2: $T_B = 2 + 8 + 9$;

n = 3: $T_B = 2 + 6 + 8 + 9$;

n = 4: $T_B = 2 + 2*6 + 8 + 9$;

n = 5: $T_B = 2 + 4 + 2*6 + 8 + 9$;

n = 8: $T_B = 2 + 4*4 + 2*6 + 8 + 9$;

n = 10: $T_B = 2*2 + 4*4 + 2*6 + 8 + 9$;

… …

Extending the above approach for any N and n,

For $2 \le n \le 2^{N-2}$,

Number of clock cycles, $CLK_B$:

$$CLK_B = 3 + n(2N - 1) \tag{3.2}$$

Number of state transitions, $T_B$:

$$\begin{aligned}
&\text{if } n=2, && T_B(2,N) = 2+2(N-1)+(2N-1)\\
&\text{if } n=3\sim 4, && T_B(3\sim 4,N) = 2+(n-2)(2(N-2))+2(N-1)+(2N-1)\\
&\text{if } n=5\sim 8, && T_B(5\sim 8,N) = 2+(n-2^2)(2(N-3))+2^2(N-2)+2(N-1)+(2N-1)\\
&\cdots \quad \cdots
\end{aligned}$$

$$\begin{aligned}
T_B &= 2 + (n - 2^l) \times 2(N - l - 1) + \sum_{j=1}^{l} \left[2^j (N - j)\right] + 2N - 1 \\
&= 2^{(l+2)} + 2n(N - l - 1) - 1
\end{aligned} \tag{3.3}$$

Where,

$$l = \lceil \log_2 n \rceil - 1 \tag{3.4}$$

For $2^{N-2} \leq n \leq 2^{N-1}$,

$$CLK_B = n(2N-1) \tag{3.5}$$

And

$$T_B = 2(n-2^l) + 2^l(N-l) + 2^{l-1}(N-(l-1)) + \cdots + 2^2(N-2) + 2(N-1) + 2N - 1$$

Therefore, Number of state transitions, $T_B$:

$$\begin{aligned} T_B &= 2(n-2^l) + 2N - 1 + \sum_{j=1}^{l} \left[2^j (N-j)\right] \\ &= 2^{(l+1)}(N-l) + 2n - 3 \end{aligned} \tag{3.6}$$

At protocol-level, power and energy consumptions are proportional to the sum of products of the number of state transitions and the Hamming distance between the states. However, by doing the following state encoding in Fig. 1: S0 = 00, S1 = 01, S2 = 10 and S3 = 11, we can guarantee that the Hamming distances for all state transitions are 1. Therefore, power and energy consumption can be estimated directly from the number of transitions. The same is also true for other protocols discussed in the rest of the work, as long as the states are properly encoded.

## 3.2. Protocol-Level Analysis of Query-Tree Protocol

As mentioned in section 3.1.1, the distribution of tags that gives the worst-case scenario will be the same as that shown in Table 3.4, except for the # of transitions and # of clock cycles. Therefore, say for N = 4, and n = 4, using Table 3.4 we can select the four tags that will give the worst-case scenario to be: Tag# 8 (or Tag# 9), Tag# 10 (or Tag# 11), Tag# 12 (or Tag# 13), and Tag# 14 (or Tag# 15).

Table 3.5 shows tag identification process for this protocol for N = 5 with 2 tags present. The table is constructed by following Figure 2.3 (State diagram for Query-Tree Protocol) shown in section 2.3. In the $4^{th}$ row of the table, under PS and NS columns, the states (S2) are multiplied by 4 and $T_{2\text{-}2}$ and Clk have a value of 4 to show that the tag remains in state S2 for four clock cycles.

From the table, we can see at the $12^{th}$ cycle upon receiving "0", Tag# 30 changes to S0 while Tag# 28 jumps to S1 (the bold font shows the state of Tag# 28). Tag# 30 then waits in S0 until Tag# 28 is identified and then jumps to S1 at the next rotation of identification process that starts at the $15^{th}$ cycle.

Before deriving number of clock cycles ($CLK_Q$) and the number of state transitions ($T_Q$) for the query-tree protocol under worst-case scenario for any N and n, first let us calculate $CLK_Q$ (n, 5)and $T_Q$ (n, 5) for N = 5 and different values of n.

We note that the number of transitions that we are interested with does not include state transitions resulted from self-loops ($T_{1\text{-}1}$ and $T_{2\text{-}2}$). Hence,

$$T_Q = T_{0-1} + T_{1-0} + T_{1-2} + T_{2-0}$$
$$= T_{QT}(n, N) - T_{1-1} - T_{2-2} \quad (3.7)$$

Where $T_{QT}(n, N)$ is the number of total state transitions including the self-loops ($T_{1-1}$ and $T_{2-2}$) in Figure 2.3.

**Table 3.5: Identification Process of Two Tags by Query-Tree Protocol for N = 5**

| Reader | PS | $T_{0-1}$ | $T_{1-0}$ | $T_{1-1}$ | $T_{1-2}$ | $T_{2-0}$ | $T_{2-2}$ | Clk | NS | Tag |
|---|---|---|---|---|---|---|---|---|---|---|
| Null | S0 | 1 | | | | | | 1 | S1 | |
| Null | S1 | | | | | | | 1 | S2 | |
| | 4*S2 | | | | | | 4 | 4 | 4*S2 | |
| | S2 | | | | | 1 | | 1 | S0 | |
| Null | S0 | 1 | | | | | | 1 | S2 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 0 | S1 | | 1 | | | | | 1 | **S1**, S0 | |
| Null | **S1**, S0 | | | | | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | | 1 | S0 | #28 |
| Null | S0 | 1 | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S2 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S2 | |
| Null | S1 | | | | 1 | | | 1 | S1 | |
| | S2 | | | | | 1 | | 1 | S3 | #30 |
| Total | - | 3 | 1 | 7 | 2 | 2 | 4 | 21 | - | - |

**Table 3.6: Commands used for Query-Tree Protocol with corresponding # of transitions and # of clock cycles for N = 4 and N = 5**

**N = 4**

| Reader Command | #of Transitions | #of Clk |
|---|---|---|
| Null-Null (N-N) | 6 | 6 |
| N-0-N | 2 | 6 |
| N-0-0-N | 2 | 6 |
| N-0-0-0-N | 2 | 6 |
| N-0-0-1-N | 2 | 6 |
| N-0-1-N | 2 | 6 |
| N-0-1-0-N | 2 | 6 |
| N-0-1-1-N | 2 | 6 |
| N-1-N | 6 | 6 |
| N-1-0-N | 3 | 6 |
| N-1-0-0-N | 3 | 6 |
| N-1-0-1-N | 3 | 6 |
| N-1-1-N | 6 | 6 |
| N-1-1-0-N | 4 | 6 |
| N-1-1-1-N | 6 | 6 |

**N = 5**

| Reader Command | #of Transitions | #of Clk |
|---|---|---|
| Null-Null (N-N) | 7 | 7 |
| N-0-N | 2 | 7 |
| N-0-0-N | 2 | 7 |
| N-0-0-0-N | 2 | 7 |
| N-0-0-1-N | 2 | 7 |
| N-0-0-0-0-N | 2 | 7 |
| N-0-0-0-1-N | 2 | 7 |
| N-0-0-1-0-N | 2 | 7 |
| N-0-0-1-1-N | 2 | 7 |
| N-0-1-N | 2 | 7 |
| N-0-1-0-N | 2 | 7 |
| N-0-1-1-N | 2 | 7 |
| N-0-1-0-0-N | 2 | 7 |
| N-0-1-0-1-N | 2 | 7 |
| N-0-1-1-0-N | 2 | 7 |
| N-0-1-1-1-N | 2 | 7 |
| N-1-N | 7 | 7 |
| N-1-0-N | 3 | 7 |
| N-1-0-0-N | 3 | 7 |
| N-1-0-1-N | 3 | 7 |
| N-1-0-0-0-N | 3 | 7 |
| N-1-0-0-1-N | 3 | 7 |
| N-1-0-1-0-N | 3 | 7 |
| N-1-0-1-1-N | 3 | 7 |
| N-1-1-N | 7 | 7 |
| N-1-1-0-N | 4 | 7 |
| N-1-1-1-N | 7 | 7 |
| N-1-1-0-0-N | 4 | 7 |
| N-1-1-0-1-N | 4 | 7 |
| N-1-1-1-0-N | 5 | 7 |
| N-1-1-1-1-N | 7 | 7 |

Table 3.6 will help us determine the type of commands the reader uses and the corresponding number of transitions ($T_{QT}$) and clock cycles during tag interrogation.

For N = 5 and different number of tags (n), the worst case tag (Tag# 30) will follow the following pattern:

- For N = 5, n = 2:

We can see from Table 3.5, the commands from the reader are: N-N, N-1-1-1-0-N, and N-1-1-1-1-N; where N stands for Null.

From Table 3.6, we see that N-N command needs 7 numbers of transitions and 7 number of clock cycles. This is evident from Table 3.5, where the read cycle for N-N command ends after state jumps from S2 to S0, at the $7^{th}$ cycle.

Hence in reference to Table 3.6,

$CLK_Q (2, 5) = 7 + 7 + 7 = 3*(5 + 2);$

$T_{QT} (2, 5) = 7 + 5 + 7 = 2*(5 + 2) + 5;$

- For N = 5, n = 3:
  - The commands used are: N-N, N-1-1-0-N, N-1-1-1-N, N-1-1-1-0-N, N-1-1-1-1-N

Hence in reference to Table 3.6,

$CLK_Q (3, 5) = 7 + 7 + 7 + 7 + 7= 5*(5 + 2);$

$T_{QT} (3, 5) = 7 + 4 + 7 + 5 + 7 = 3*(5 + 2) + (5 - 1) + 5;$

- For N = 5, n = 4:
  - The commands used are: N-N, N-1-1-0-N, N-1-1-1-N, N-1-1-0-0-N, N-1-1-0-1-N, N-1-1-1-0-N, N-1-1-1-1-N

Hence in reference to Table 3.6,

$CLK_Q(4, 5) = 7 + 7 + 7 + 7 + 7 + 7 + 7 = 7*(5 + 2);$

$T_{QT}(4, 5) = 7 + 4 + 7 + 4 + 4 + 5 + 7 = 3*(5 + 2) + 3*(5 - 1) + 5;$

Using similar approach, for N = 5 and for the rest of the following tag distributions (n):

n = 5: $CLK_Q(5, 5) = 9*(5 + 2);$ $T_{QT} = 4*(5+2)+(5-2)+3*(5-1)+5;$

n = 6: $CLK_Q(6, 5) = 11*(5 + 2);$ $T_{QT} = 4*(5+2)+3*(5-2)+3*(5-1)+5;$

n = 7: $CLK_Q(7, 5) = 13*(5 + 2);$ $T_{QT} = 4*(5+2)+5*(5-2)+3*(5-1)+5;$

n = 8: $CLK_Q(8, 5) = 15*(5 + 2);$ $T_{QT} = 4*(5+2)+7*(5-2)+3*(5-1)+5;$

n = 9: $CLK_Q(9, 5) = 17*(5 + 2);$ $T_{QT} = 5*(5+2)+(5-3)+7*(5-2)+3*(5-1)+5;$

n = 10: $CLK_Q(10, 5) = 19*(5 + 2);$ $T_{QT} = 5*(5+2)+3*(5-3)+7*(5-2)+3*(5-1)+5;$

… …

After considering different Ns (N = 3, 4, and 6) for different ns, the following general equation is deduced:

For $2 \leq n \leq 2^{N-1}$,

Number of clock cycles, $CLK_Q$:

$$CLK_Q = (2n-1)(N+2) \tag{3.8}$$

Number of state transitions, $T_{QT}$:

$if\ n=2,\quad T_{QT}(2,N) = 2(N+2)+N$

$if\ n=3\sim 4,\quad T_{QT}(3\sim 4,N) = 3(N+2)+[2(n-2)-1](N-1)+N$

$if\ n=5\sim 8,\quad T_{QT}(5\sim 8,N) = 4(N+2)+[2(n-4)-1](N-2)+3(N-1)+N$

… …

$$T_{QT} = T_{QT}(n, N) = (l+2)(N+2) + \left[2\left(n - 2^l\right) - 1\right](N-l) + \sum_{j=1}^{l}\left[\left(2^j - 1\right)(N - j + 1)\right]$$
$$= \left[2\left(n - 2^l\right) - 1\right](N-l) + 2^{(l+1)}(N - l + 2) + \frac{l^2 + 3l}{2} \qquad (3.9)$$

Where l is given by (3.4).

For the self-loops, after performing similar analysis we obtained the following results:

For N = 5:

n = 2: $T_{1\text{-}1} = (5\text{-}2) + (5\text{-}1)$

n = 3: $T_{1\text{-}1} = (5\text{-}3) + 2*(5\text{-}2) + (5\text{-}1)$

n = 4: $T_{1\text{-}1} = 3*(5\text{-}3) + 2*(5\text{-}2) + (5\text{-}1)$

n = 5: $T_{1\text{-}1} = (5\text{-}4) + 4*(5\text{-}3) + 2*(5\text{-}2) + (5\text{-}1)$

… …

And

n = 2: $T_{2\text{-}2} = (5\text{-}1)$

n = 3: $T_{2\text{-}2} = (5\text{-}1) + 1$

n = 4: $T_{2\text{-}2} = (5\text{-}1) + 1$

n = 5: $T_{2\text{-}2} = (5\text{-}1) + 1 + 2$

… …

Therefore for any N and n, where $2 \le n \le 2^{N-1}$,

*if* $n = 2$, $T_{1\text{-}1}(2, N) = (N-2) + (N-1)$

*if* $n = 3 \sim 4$, $T_{1\text{-}1}(3 \sim 4, N) = \left[2\left(n - 2^l\right) - 1\right](N-3) + 2(N-2) + (N-1)$

… …

$$T_{1-1} = \left[2\left(n-2^{l}\right)-1\right]\left[N-(l+2)\right]+\sum_{j=0}^{l}\left\{2^{l-j}\left[N-(l+1-j)\right]\right\}$$
$$= 2n(N-l-2)+2^{(l+2)}-2N+l+1 \qquad (3.10)$$

*if* $n=2$, $T_{2-2}(2,N)=(N-1)$
*if* $n=3\sim4$, $T_{2-2}(3\sim4,N)=(N-1)+1$
... ...

$$T_{2-2} = (N-1)+\sum_{j=1}^{l} j \quad = (N-1)+l\left(\frac{l+1}{2}\right) \qquad (3.11)$$

Therefore, the number of state transitions ($T_Q$) will be:

$$T_Q = T_{QT} - T_{1-1} - T_{2-2} = l+4n \qquad (3.12)$$

For instance, for N = 5 and n = 8, we have $CLK_Q$ = 15*7 =105 and $T_Q$ = 2 + 4*8 = 34

## 3.3. Protocol-Level Analysis of Improved Query-Tree Protocol

Table 3.5, used for query-tree protocol, can be modified for this protocol except that there will be a "Stop Send" (or an "SS") command from the reader at the start of the 4th cycle. Table 3.7 shows tag identification process for this protocol for N = 5 with 2 tags present. Even though the reduction of clock cycles is not evident through the comparison of Table 3.5 and Table 3.7, as the number of tags increases, the issuing of "SS" command further decreases the number of total clock cycles. For example for N = 5: if n = 4, $CLK_Q$ = 49 and $CLK_{IQ}$ = 48 but if n = 16, $CLK_Q$ = 217 and $CLK_{IQ}$ = 206. For N = 7: if n = 16, $CLK_Q$ = 279 and $CLK_{IQ}$ = 268 but if n = 64, $CLK_Q$ = 1143 and $CLK_{IQ}$ = 1086.

**Table 3.7: Identification Process of Two Tags by Improved Query-Tree Protocol for N = 5**

| Reader | PS | $T_{0-1}$ | $T_{1-0}$ | $T_{1-1}$ | $T_{1-2}$ | $T_{2-0}$ | $T_{2-2}$ | Clk | NS | Tag |
|---|---|---|---|---|---|---|---|---|---|---|
| Null | S0 | 1 | | | | | | 1 | S1 | |
| Null | S1 | | | | 1 | | | 1 | S2 | |
| | 4*S2 | | | | | | 4 | 4*S2 | S2 | |
| SS | S2 | | | | | 1 | | 1 | S0 | |
| Null | S0 | 1 | | | | | | 1 | S2 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 0 | S1 | | 1 | | | | | 1 | **S1**,S0 | |
| Null | **S1**,S0 | | | | | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | | 1 | S0 | #28 |
| Null | S0 | 1 | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| Null | S1 | | | | 1 | | | 1 | S2 | |
| | S2 | | | | | 1 | | 1 | S0 | #30 |
| Total | - | 3 | 1 | 7 | 2 | 2 | 4 | 21 | - | - |

Before deriving number of clock cycles ($CLK_{IQ}$) and the number of state transitions ($T_{IQ}$) for the improved query-tree protocol under worst-case scenario for any N and n, let us observe the following points:

- Since improved query-tree protocol is similar to query-tree protocol with the exception of different performance in state S2; and since the numbers of transitions for both protocols do not include $T_{2-2}$, the number of transitions for both cases stays the same.

Hence, the number of state transitions ($T_{IQ}$) will be:

$$T_{IQ} = T_Q = l + 4n \tag{3.13}$$

- Before deriving the number of clock cycles ($CLK_{IQ}$), let us see the pattern of total clock cycles for N = 5 and different number of tags (n):

  - For n = 2, as shown in Table 3.7, the existence of "SS" after N-N command, won't alter the number of clock cycles and stays to be 7 (as compared to that the query-tree protocol. Hence,
    - $CLK_{IQ}$ (2, 5) = 7 + 7 + 7 = 3*(5 + 2);

  - For n = 3, the existence of "SS" after N-N command decreases the clock cycle in N-N read cycle from 7 to 6 (Table similar to Table 3.7 can be drawn to see the effect). Hence,
    - $CLK_{IQ}$ (3, 5) = 6 + 7 + 7 + 7 + 7 = (5 + 1) + 4*(5 + 2);

  - For n = 4, as shown in Table 3.8, even if "SS" occurs after N-N, N-1-1-0-N, and N-1-1-1-N commands, it is only the existence of "SS" after N-N command that decreases the clock cycle in N-N read cycle from 7 to 6. Hence,
    - $CLK_{IQ}$ (4, 5) = 6 + 7 + 7 + 7 + 7 + 7 + 7 = (5 + 1) + 6*(5 + 2);

  - Similarly, for n = 8, "SS" occurs after N-N, N-1-0-N, N-1-0-0-N, N-1-1-N, N-1-1-0-N, and N-1-1-1-N commands. The existence of "SS" after N-N command decreases the clock cycle to 5; "SS" after N-1-0-N and N-1-1-N commands both decrease the clock cycle to 6. Hence,
    - $CLK_{IQ}$ (8, 5) = 5 + 2*(5 + 1) + 6*(5 + 2);

**Table 3.8: Identification Process of Four Tags by Improved Query-Tree Protocol for N = 5**

| Reader | PS | $T_{0-1}$ | $T_{1-0}$ | $T_{1-1}$ | $T_{1-2}$ | $T_{2-0}$ | $T_{2-2}$ | Clk | NS | Tag |
|---|---|---|---|---|---|---|---|---|---|---|
| Null | S0 | 1 | | | | | | 1 | S1 | |
| Null | S1 | | | | 1 | | | 1 | S2 | |
| | S2 | | | | | | 3 | 3 | S2 | |
| SS | S2 | | | | | 1 | | 1 | S0 | |
| Null | S0 | 1 | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 0 | S1 | | 1 | | | | | 1 | **S1**, S0 | |
| Null | **S1**, S0 | | | | | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | | 1 | **S2**, S0 | |
| SS | **S2**, S0 | | | | | 1 | | 1 | S0 | |
| Null | S0 | 1 | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| Null | S1 | | | | 1 | | | 1 | S2 | |
| | S2 | | | | | | 1 | 1 | S2 | |
| SS | S2 | | | | | 1 | | 1 | S0 | |
| Null | S0 | 1 | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 0 | **S1**, S0 | | 1 | | | | | 1 | **S1**, S0 | |
| 0 | **S1**, S0 | | | | | | | 1 | **S1**, S0 | |
| Null | S1 | | | | | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | | 1 | S0 | #24 |
| Null | S0 | 1 | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 0 | S1 | | 1 | | | | | 1 | **S1**, S0 | |
| 1 | **S1**, S0 | | | | | | | 1 | **S1**, S0 | |
| Null | **S1**, S0 | | | | | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | | 1 | S0 | #26 |
| Null | S0 | 1 | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 0 | S1 | | 1 | | | | | 1 | **S1**, S0 | |
| Null | **S1**, S0 | | | | | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | | 1 | S0 | #28 |
| Null | S0 | 1 | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | 1 | S1 | |
| N | S1 | | | | 1 | | | 1 | S2 | |
| | S2 | | | | | 1 | | 1 | S0 | #30 |
| Null | | 7 | 4 | 16 | 3 | 4 | 4 | 48 | | |

After performing similar analysis, the following results were obtained for N = 5:

$if\ n=2,\quad CLK_{IQ}(2,N)=3*(5+2)$
$if\ n=3,\quad CLK_{IQ}(3,N)=(5+1)+4*(5+2)$
$if\ n=4,\quad CLK_{IQ}(4,N)=(5+1)+6*(5+2)$
$if\ n=5,\quad CLK_{IQ}(5,N)=5+(5+1)+7*(5+2)$
$if\ n=8,\quad CLK_{IQ}(8,N)=5+2*(5+1)+12*(5+2)$
$if\ n=16,\quad CLK_{IQ}(16,N)=(5-1)+2*5+4*(5+1)+24*(5+2)$
... ...
*Where,* $N=5$

After considering different Ns (N = 3, 4, and 6) for different ns, the following general equation is deduced:

For any N:

For $2 \le n \le 2^{N-1}$,

$CLK_{IQ} = CLK_{IQ1} + CLK_{IQ2}$

Where $CLK_{IQ2}$ is the last term in the $CLK_{IQ}$ equation and $CLK_{IQ1}$ is the rest of the terms.

$CLK_{IQ2}$ and $CLK_{IQ1}$ are therefore,

$$\begin{aligned} CLK_{IQ2} &= 3(N+2)+2\sum_{j=1}^{k}(N+2)+\sum_{j=1}^{n-3-k}(N+2) \\ &= (N+2)[1+n+k] \end{aligned} \tag{3.14}$$

And

$$CLK_{IQ1} = \sum_{j=0}^{l-1}\left[\left(\text{int}\left(\frac{n-2^{(j+1)}-1}{2^{(j+2)}}\right)+1\right)(N+1-j)\right] \tag{3.15}$$

$$CLK_{IQ} = CLK_{IQ1} + CLK_{IQ2}$$
$$= \left\{ \sum_{j=0}^{l-1} \left[ \left( \text{int}\left( \frac{n-2^{(j+1)}-1}{2^{(j+2)}} \right) + 1 \right) (N+1-j) \right] \right\} + (N+2)[1+n+k] \tag{3.16}$$

Where l is given by (3.4), and k by (3.17)

$$k = \left\lceil \frac{n-3}{2} \right\rceil \tag{3.17}$$

As a special case, if $n = 2^m$, then

$$CLK_{IQ}(n = 2^m) = 3(2^{(m-1)})(N+2) + \sum_{n=1}^{m-1} \left[ (2^{m-(n+1)} - 1)(N+2-n) \right]$$
$$= 2^{(m-1)}(4N+6) + (m-N-1) \tag{3.18}$$

# Chapter 4 A Combined Binary-Query-Tree Protocol-The Proposed Protocol

In this section we propose a new protocol that combines the binary-tree and query-tree protocols in order to reduce the number of state transitions and clock cycles under a worst-case scenario. Figure 4.1 shows the state diagram of the proposed protocol.

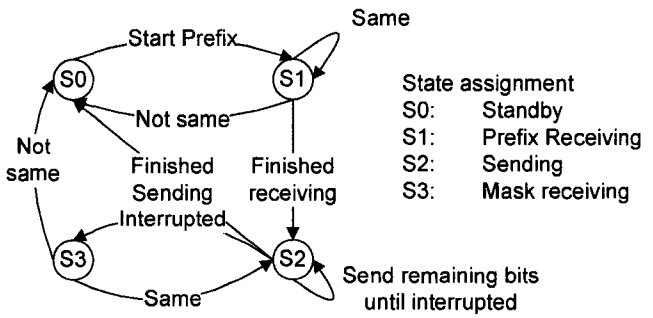


**Figure 4.1: State diagram for the Combined-Binary-Query-Tree Protocol.**

In this protocol, when a tag is in S2 and a collision is detected, it will go to another state, S3 (Mask receiving state), to receive a masking bit sent by the reader, instead of jumping to S0 (as in case of the improved query-tree protocol). Table 4.1 shows tag identification process for the proposed protocol for N = 5 with 2 tags present and Table 4.2 for N = 5 with n = 4. In both tables, bold fonts used for states, indicate the status of the states in other tags while the worst-case tag (Tag# 30) is in state "S0".

The benefit from this extra state is a reduction in both the numbers of state transitions and clock cycles. This can be evident if we examine two tags that have same first bits as the prefix bits sent by the reader and differ after the first prefix bits. In the improved query-tree protocol, the reader has to send two distinct prefix bits to read both tags.

As a result, the worst-case tag has to jump from S0 to S1 to S2 twice. In the proposed protocol, one of the tags will be identified by the masking bit (sent in S3), and the other by sending prefix bits only once. Thus, the worst-case tag will need less number of transitions and clock cycles for identification. For instance, for N = 5, n = 2, under the worst-case scenario, the improved query protocol uses 8 state transitions and 21 clock cycles; whereas this protocol uses 7 state transitions and 15 clock cycles.

**Table 4.1: Identification Process of Two Tags by Combined Binary-Query-Tree Protocol for N = 5**

| Reader | PS | $T_{0-1}$ | $T_{1-0}$ | $T_{1-1}$ | $T_{1-2}$ | $T_{2-0}$ | $T_{2-2}$ | $T_{2-3}$ | $T_{3-2}$ | $T_{3-0}$ | Clk | NS | Tag |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Null | S0 | 1 | | | | | | | | | 1 | S1 | |
| Null | S1 | | | | 1 | | | | | | 1 | S2 | |
| | S2 | | | | | | 3 | | | | 3 | S2 | |
| SS | S2 | | | | | | | 1 | | | 1 | S3 | |
| 0 | S3 | | | | | | | | | 1 | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | | | | | 1 | S0 | #28 |
| Null | S0 | 1 | | | | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | | | | 1 | S1 | |
| Null | S1 | | | | 1 | | | | | | 1 | S2 | |
| | S2 | | | | | 1 | | | | | 1 | S0 | #30 |
| Total | | 2 | 0 | 4 | 2 | 1 | 3 | 1 | 0 | 1 | 15 | | |

This protocol, however, needs an extra hardware in order to decrement its ID pointer while in S3. This is necessary because the tag receives a masking bit in S3 that needs to be compared with the previously sent bit.

By following the similar steps in deriving the number of clock cycles for the query-tree protocol, we can find the number of state transitions and number of clock cycles for the combined binary-query-tree protocol (again under worst-case scenario) as follows:

$$\begin{aligned} T_{BQ} &= \sum (T_{0-1} + T_{1-0} + T_{1-2} + T_{2-0} + T_{2-3} + T_{3-2} + T_{3-0}) \\ &= T_{BQT} - T_{1-1} - T_{2-2} \end{aligned} \tag{4.1}$$

Where $T_{BQT}(n, N)$ is the number of total state transitions including the self-loops ($T_{1-1}$ and $T_{2-2}$) in Figure 4.1.

**Table 4.2: Identification Process of Four Tags by Combined Binary-Query-Tree Protocol for N = 5**

| Reader | PS | $T_{0-1}$ | $T_{1-0}$ | $T_{1-1}$ | $T_{1-2}$ | $T_{2-0}$ | $T_{2-2}$ | $T_{2-3}$ | $T_{3-2}$ | $T_{3-0}$ | Clk | NS | Tag |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Null | S0 | 1 | | | | | | | | | 1 | S1 | |
| Null | S1 | | | | 1 | | | | | | 1 | S2 | |
| | S2 | | | | | | 2 | | | | 2 | S2 | |
| SS | S2 | | | | | | | 1 | | | 1 | S3 | |
| 0 | S3 | | | | | | | | | 1 | 1 | **S2**, S0 | |
| SS | **S2**, S0 | | | | | | | | | | 1 | **S3**, S0 | |
| 0 | **S3**, S0 | | | | | | | | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | | | | | 1 | S0 | #24 |
| Null | S0 | 1 | | | | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | | | | 1 | S1 | |
| 0 | S1 | | 1 | | | | | | | | 1 | **S1**, S0 | |
| 1 | **S1**, S0 | | | | | | | | | | 1 | **S1**, S0 | |
| Null | **S1**, S0 | | | | | | | | | | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | | | | | 1 | S0 | #26 |
| Null | S0 | 1 | | | | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | | | | 1 | S1 | |
| Null | S1 | | | | 1 | | | | | | 1 | S2 | |
| SS | S2 | | | | | | | 1 | | | 1 | S3 | |
| 0 | S3 | | | | | | | | | 1 | 1 | **S2**, S0 | |
| | **S2**, S0 | | | | | | | | | | 1 | S0 | #28 |
| Null | S0 | 1 | | | | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | | | | 1 | S1 | |
| 1 | S1 | | | 1 | | | | | | | 1 | S1 | |
| Null | S1 | | | | 1 | | | | | | 1 | S2 | |
| | S2 | | | | | 1 | | | | | 1 | S0 | #30 |
| Total | | 4 | 1 | 9 | 3 | 1 | 2 | 2 | 0 | 2 | 31 | | |

For N = 5 and different number of tags (n), the worst case tag (Tag# 30) will follow the following pattern:

- For N = 5, n = 2:

We can see from Table 4.1, the commands from the reader are: N-N-0 and N-1-1-1-1-N; where N stands for Null. Here the intermediate command "SS" is left out from N-N-0 command, but it is assumed to be within N-N-0 command. The same is true in the subsequent derivations.

From Table 4.1, we see that N-N-0 command needs 7 numbers of transitions and 8 number of clock cycles. This is evident from Table 4.1, where the read cycle for N-N-0 command ends after reading Tag# 28, at the $8^{th}$ cycle. Hence,

$CLK_{BQ}(2, 5) = 8 + 7;$

$T_{BQT}(2, 5) = 7 + 7;$

- For N = 5, n = 3:

  - The commands used are: N-N-0, N-1-1-N-0, and N-1-1-1-1-N. Hence,

$CLK_{BQ}(3, 5) = 8 + 8 + 7;$

$T_{BQT}(3, 5) = 6 + 7 + 7;$

- For N = 5, n = 4:

  - The commands used are (from Table 4.2): N-N-0-0, N-1-1-0-1-N, N-1-1-N-0, and N-1-1-1-1-N. Hence,

$CLK_{BQ}(4, 5) = 9 + 7 + 8 + 7;$

$T_{BQT}(4, 5) = 6 + 4 + 7 + 7;$

Using similar approach, for N = 5 and for the rest of the following tag distributions (n), the following pattern is developed:

Number of clock cycles:

n=2: $CLK_{BQ} = 2*(5+3)-1$

n=3: $CLK_{BQ} = 2*(5+3)-1$

n=8: $CLK_{BQ} = 2*(5+3)-1$

Number of state transitions:

n=2: $T_{BQT} = 2*(5+2);$

n=3: $T_{BQT} = (5+1) + 2*(5+2);$

n=4: $T_{BQT} = (5-1) + (5+1) + 2*(5+2);$

n=5: $T_{BQT} = (5-1) + 5 + (5+1) + 2*(5+2);$

n=6: $T_{BQT} = (5-2) + (5-1) + 5 + (5+1) + 2*(5+2);$

n=7: $T_{BQT} = 2*(5-2) + (5-1) + 5 + (5+1) + 2*(5+2);$

n=8: $T_{BQT} = 3*(5-2) + (5-1) + 5 + (5+1) + 2*(5+2);$

n=9: $T_{BQT} = 3*(5-2) + (5-1) + (5-1) + 5 + (5+1) + 2*(5+2);$

n=10: $T_{BQT} = (5-3) + 3*(5-2) + (5-1) + (5-1) + 5 + (5+1) + 2*(5+2);$

. . .

n=16: $T_{BQT} = (16-2^3-1)*(5-3) + 3*(5-2) + (5-1) + (5-1) + 5 + (5+1) + 2*(5+2);$

After considering different Ns (N = 3, 4, and 6) for different ns, the following general equation is derived:

For $2 \leq n \leq 2^{N-1}$,

$$CLK_{BQ} = n(N+3)-1 \tag{4.2}$$

And

$$\begin{aligned} T_{BQT} &= (n-2^l-1)(N-l)+\sum_{j=1}^{l-1}[(2^j-1)(N-j)]+\sum_{j=1}^{l}(N+2-j)+2(N+2) \\ &= 2^{(l+1)}+l(2-n)+Nn+2 \end{aligned} \tag{4.3}$$

Where l is again given by (3.4).

For the self-loops, we have

For N = 5:

n=2: $T_{1\text{-}1} = (5\text{-}1)$

n=3: $T_{1\text{-}1} = (5\text{-}2) + (5\text{-}1)$

n=4: $T_{1\text{-}1} = (5\text{-}3) + (5\text{-}2) + (5\text{-}1)$

n=5: $T_{1\text{-}1} = 2*(5\text{-}3) + (5\text{-}2) + (5\text{-}1)$

n=6: $T_{1\text{-}1} = (5\text{-}4) + 2*(5\text{-}3) + (5\text{-}2) + (5\text{-}1)$

n=8: $T_{1\text{-}1} = 3*(5\text{-}4) + 2*(5\text{-}3) + (5\text{-}2) + (5\text{-}1)$

… …

And

n=2: $T_{2\text{-}2} = (5\text{-}2)$

n=3: $T_{2\text{-}2} = (5\text{-}3)$

n=4: $T_{2\text{-}2} = (5\text{-}3)$

n=5: $T_{2\text{-}2} = (5\text{-}4)$

n=8: $T_{2\text{-}2} = (5\text{-}4)$

… …

Therefore for any N and n, where $2 \le n \le 2^{N-1}$,

$$\begin{aligned}
&if\ n=2, \quad && T_{1-1}(2,N)=(N-1)\\
&if\ n=3\sim 4, && T_{1-1}(3\sim 4,N)=(n-3)(N-3)+(N-2)+(N-1)\\
&\cdots \quad \cdots
\end{aligned}$$

$$\begin{aligned}
T_{1-1} &= \left(n-2^{l}-1\right)\left[N-(l+2)\right]+\sum_{j=0}^{l-1}\left\{2^{l}\left[N-j-2\right]\right\}+(N-1)\\
&= n(N-l-2)+2^{(l+1)}-N+l+1
\end{aligned} \tag{4.4}$$

And

$$T_{2-2} = (N-l)-2 \tag{4.5}$$

Therefore,

$$T_{BQ} = T_{BQT} - T_{1-1} - T_{2-2} = 2(l+n)+3 \tag{4.6}$$

It should be mentioned that the states in Figure 4.1 can be encoded optimally as follows: S0 = 00, S1 = 01, S2 = 11 and S3 = 10. While the Hamming distance between S0 and S2 is 2, the number of transitions between them is only 1. Therefore, the power consumption of this protocol can still reasonably be estimated from the number of its state transitions.

# Chapter 5 Protocol-Level Comparison of the Four Tree Protocols

## 5.1. Protocol-level Comparison of the Binary, Query, and Improved Query Tree Protocols

In this section, we evaluate the protocol level performance of the above three different protocols in terms of total number of state transitions, number of clock cycles, energy and power consumption, all under their worst cases. The number of state transitions is a key metric for estimation of power and energy consumption, while the number of clock cycles determines how fast the tags can be identified. Their ratio represents the transitions per clock cycle which measures energy consumption.

In order to compare their power consumption in a fair manner, the three protocols are assumed to have same latency (i.e., the tags are assumed to be identified within a same time duration for different protocols). Thus, a particular protocol with less clock cycles can use a lower clock frequency to achieve power reduction. We choose the improved query-tree protocol as a reference, and the number of clock cycles for other two protocols is normalized to obtain an equivalent value as follows:

$$\left.\begin{aligned} T_{B,eq} &= T_B\left(\frac{CLK_B}{CLK_{IQ}}\right) \\ T_{Q,eq} &= T_Q\left(\frac{CLK_Q}{CLK_{IQ}}\right) \\ T_{IQ,eq} &= T_{IQ} \end{aligned}\right\} \qquad (5.1)$$

Thus, power consumption of the binary-tree, query-tree and improved query-tree protocols is proportional to, and measured by, $T_{B,\,eq}$, $T_{Q,\,eq}$ and $T_{IQ,\,eq}$, respectively.

Comparison of the three protocols is shown in Figure 5.1 (for n = 4) and in Figure 5.2 (for n = 8) which is plotted using equations (3.5) ~ (3.8), (3.12) ~ (3.17), and (5.1). From both figures, it can be seen that the query-tree and improved query-tree protocols are preferable to binary-tree protocol in terms of number of transitions, and energy. Regarding power consumptions, by examining the two figures we can deduce that there is a region where the binary-tree is preferable over either of the two query-trees or vice versa. The point that divides the two regions is a function of N and n. For instance, for n = 4, Figure 5.1 shows that improved query protocol is better than binary-tree for any N. However, for n = 8, Figure 5.2 shows that the binary-tree will consume less power than the rest of the two protocols if N is less than 5.

From simulation, we can find the region where the binary-tree is preferable over the two tree types or vice versa in terms of power consumption. Table 5.1 is the result of this simulation that gives the threshold value of n ($n_{th}$) for every N that defines this region. Hence, for a particular N, if the number of tags (n) is greater or equal to $n_{th}$ then the binary-tree protocol will consume less power. For example, for N = 5 if $n \geq 9$, the binary-tree is preferable over the two query-tree types.

In the region where $n < n_{th}$, the improved query-tree has relatively lower power consumption than the query-tree, even though this is not evident from the two figures. For instance, for N = 5 if n = 8, $T_{IQ, eq}$, = 34 and $T_{Q, eq}$, = 35; and for N = 6 if n = 16, $T_{IQ, eq}$, = 67 and $T_{Q, eq}$, = 70.

Moreover the binary-tree uses lower clock cycles than the rest as shown by Figure 5.1 and Figure 5.2.

**Table 5.1: Tabulation of $n_{th}$ for every N that determines the region $n \geq n_{th}$ where the binary-tree has lower power consumption over the two query-tree types**

| N | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|
| $n_{th}$ | 5 | 9 | 17 | 38 | 81 | 171 | 357 | 738 | 1515 |

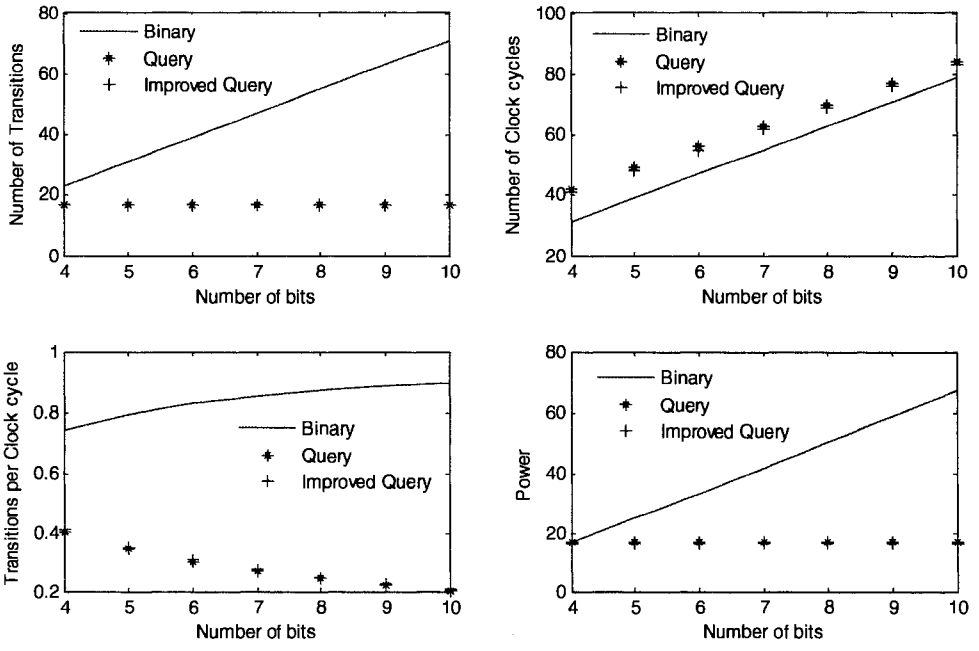


**Figure 5.1: Performance comparison of binary-tree, query-tree and improved-query-tree protocols under their worst cases (n = 4)**

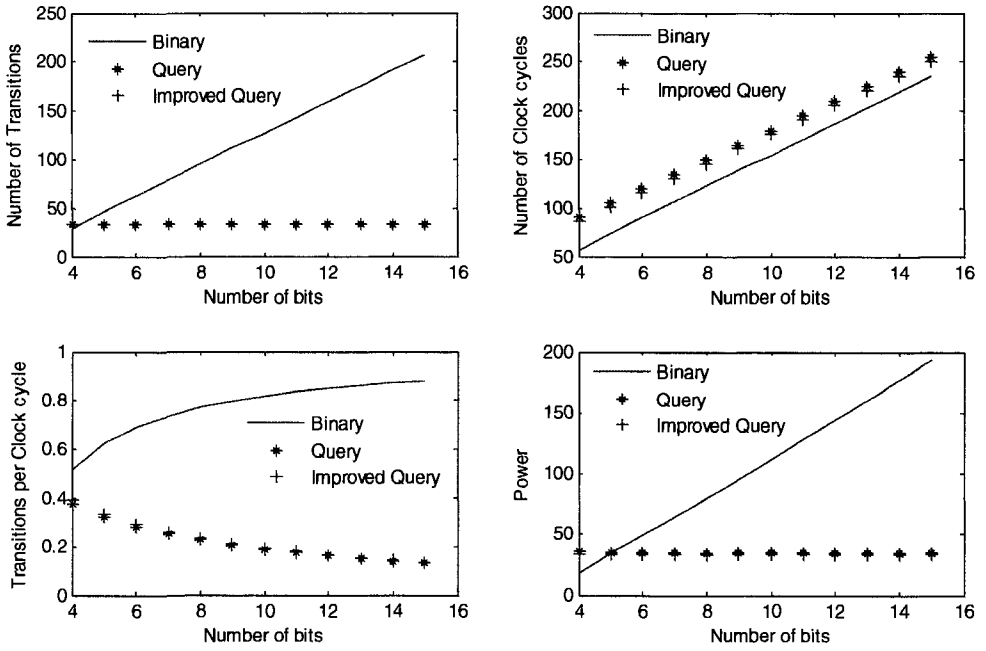


**Figure 5.2: Performance comparison of binary-tree, query-tree and improved-query-tree protocols under their worst cases (n = 8)**

## 5.2. Protocol-Level Comparison of the Proposed Protocol with Binary and Improved Query Protocols

Equation (4.6) is modified to determine the equivalent value of the total state transitions for the proposed protocol (we still chose the improved query-tree protocol as a reference). The resulted equation is then,

$$T_{BQ,eq} = T_{BQ}\left(\frac{CLK_{BQ}}{CLK_{IQ}}\right) \tag{5.2}$$

Figure 5.3 is plotted using equations (3.5), (3.6), (3.13), (3.16), (4.2), (4.6), (5.1), and (5.2) to show the comparison of the proposed protocol with the binary-tree and improved query-tree protocols. The graph is plotted for tag distributions (n) = 4. It can be seen from the figure that the proposed protocol outperforms the other two protocols in terms of number of transitions, number of clock cycles and power. However, from the energy consumption point of view, the improved query-tree protocol shows the best performance. Unlike the previous comparison (the comparison between the binary-tree and the two query-tree protocols) where the power consumption was dependant on $n_{th}$, here the proposed protocol always consumes less power over the rest of the protocols.

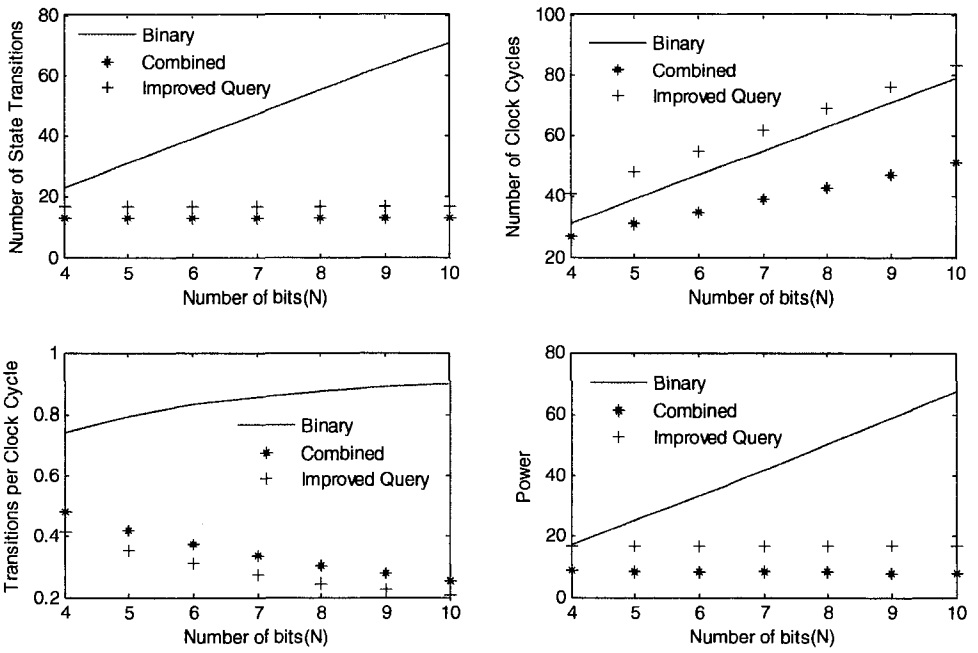


**Figure 5.3: Performance comparison of binary-tree, improved query-tree and combined binary-query-tree protocols under their worst cases (n = 4).**

# Chapter 6 Implementation and Comparison of the Four Protocols at Different Levels of Abstraction

Implementation of the Binary-Tree protocol, Query-Tree Protocol, Improved Query-Tree Protocol, and Combined-Binary-Tree Protocol, for a tag with N = 4, at different levels of abstraction namely, RTL level, Gate level and layout level is performed and the corresponding results are shown accordingly. For all protocols, the design was based on the worst-case scenario; that is Tag# 15 (1111) was selected for implementation. Hence, the following functional specifications are followed to design the logic:

The ID is represented by a 4 bit number, and Tag# 15 = 1111 is used for implementation; hence no memory is needed to store Tag# 15.

Inputs to the tag protocol circuit are

- internal Reset
- signals coming from the reader: Start (or Null), Clock, Input_R (enquiring bit)
- Output from the tag protocol circuit is Out1 with the mapping showing in Table 6.1:

**Table 6.1: Mapping of Out1 and Out2 to the tag response**

| Response from the tag | Out1 | Out2 |
|---|---|---|
| No response | 0 | 0 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |

The output in our design (for Tag# 15) only uses Out1, because its ID = 1111, and it either responds a "No response" when it is in state "S0" or a bit from its ID (which is always "1"). Therefore, Out2 is not needed. But if a tag with an ID containing a "0" (for example Tag# 13 = 1101), we definitely need to have both Out1 and Out2.

## 6.1. Implementation and Simulation of the Binary-Tree Protocol

Using Verilog-XL, an RTL Verilog code for Tag# 15 was written. For n = 4 (number of tags to be read = 4) and for the worst case scenario, a test bench was written in Verilog in order to verify the functionality of the written Verilog code. The functionality of the HDL code was verified using SimVision and the resulting waveform is shown in Figure 6.1.

In Figure 6.1: ID_byte [3:0] = F = 1111 corresponds to Tag# 15, Input_R shows the enquiry bit coming from the reader, Null is a control signal from the reader used for starting from S0 to S1, Out1 indicates the response from the tag, Reset is used to bring the tag to the active state (power up state), S0 ~ S 3 show the assigned values for the four states and state[1:0] indicates the status of the state at different times, and index[1:0] is used to point the current ID_byte bit location.

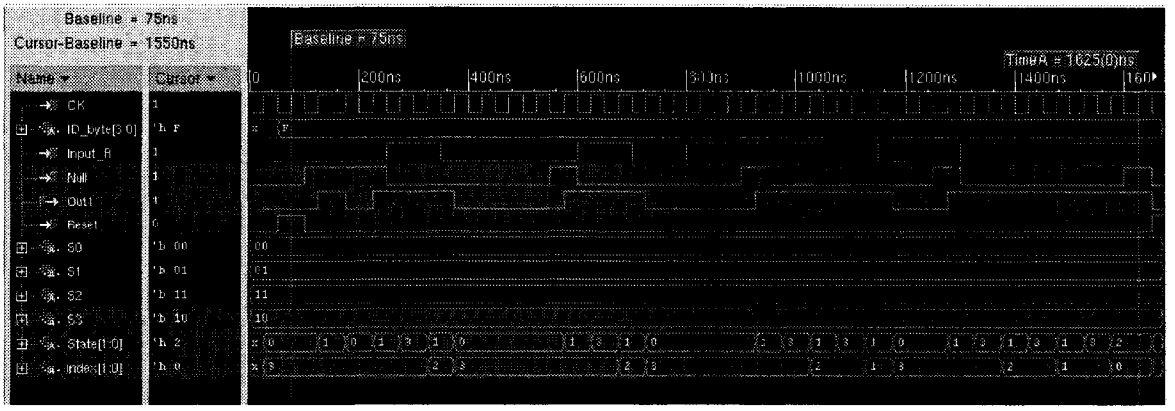


**Figure 6.1: Simulation output for Tag # 15 (N = 4, n = 4): Binary-Tree Protocol.**

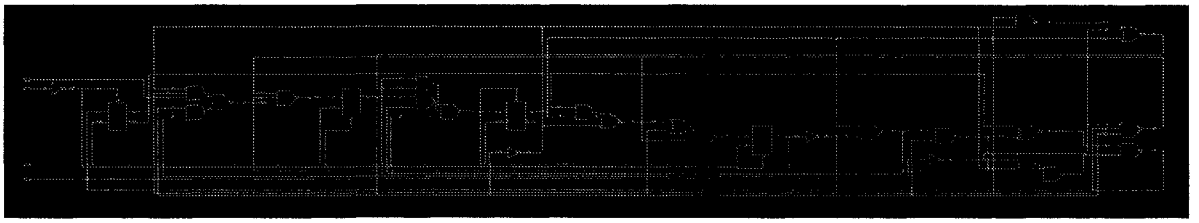

**Figure 6.2: Schematic for Tag # 15 (N = 4): Binary-Tree Protocol**

As can be seen from Figure 6.1, the operating frequency is 20MHz and it takes 1550ns for Tag# 15 to be read; hence the total number of clock cycles = 1550ns * 20MHz = 31. The reading starts at 75ns due to 'Reset' where the state is at S0. At 125ns, the state jumps to S1 since Null (or Start) = 1. But at the next positive edge of the clock (at 175ns) the state returns to S0 since the enquiry bit from reader = 0 and is different from the msb of the ID_byte which is 1, as a result, the tag signals a no-response situation by sending Out1 = 0. Through successive enquiry-response pattern that resembles Table 1 (except for N = 4, n= 4) the reading continues and finally ends at 1625ns (where the tag is killed at S3).

The HDL code was synthesized using design_vision from Synopsys and the resulting schematic is shown in Figure 6.2.

Using Encounter and Virtuoso from Cadence, the layout was designed and the final result is shown in Figure 6.3.

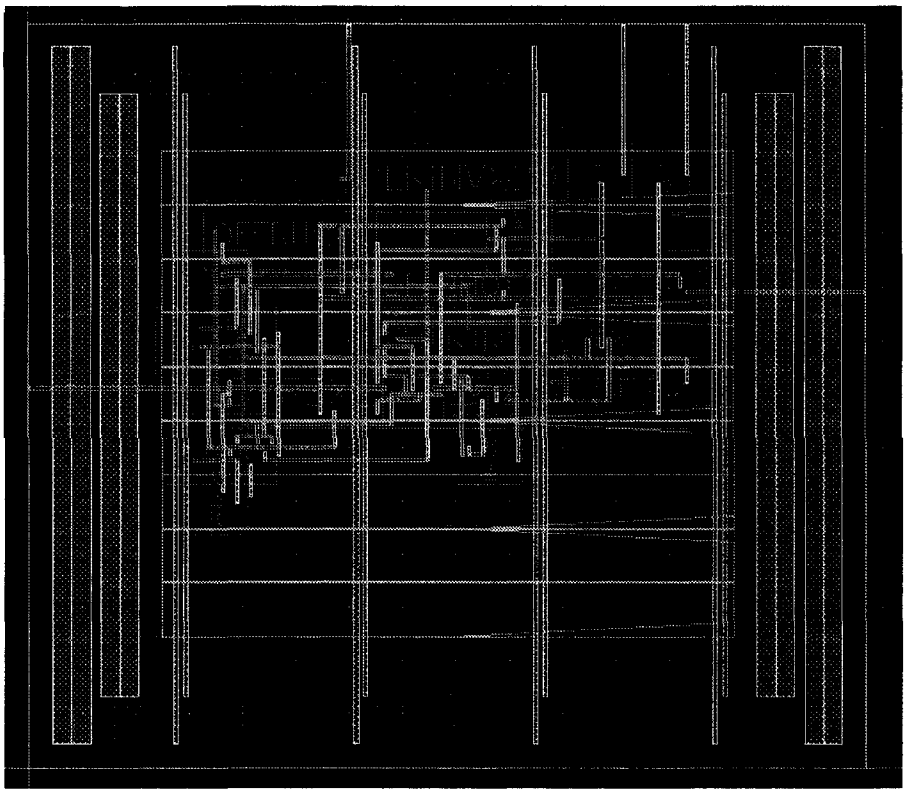

**Figure 6.3: Layout for Tag # 15 (N = 4): Binary-Tree Protocol**

## 6.2. Implementation and Simulation of the Query-Tree Protocol

After implementing Tag# 15 at different levels of abstraction using the Query-Tree Protocol, the corresponding results are shown in Figure 6.4, 6.5 and 6.6. The RTL Verilog code for this protocol was written based on Figure 2.3, but with the following additional information: the condition for transition from state "S0" to "S1" and from "S1" to "S2" is when "Null" = "1" and "Input_R" = "0". Appendix A.2.1 shows the full code.

Figure 6.4 shows the verification of the functionality of the designed Verilog code. Figure 6.5 shows the schematic that was obtained after synthesizing the Verilog code using design_vision and Figure 6.6 shows the final layout from Virtuoso.

From Figure 6.4, that shows the simulation result for the Verilog top-level module using SimVision, we can see that it takes 2100ns for Tag# 15 to be read. Hence, the total number of clock cycles = 2100ns * 20MHz = 42 clock cycles.

The waveform shown in Figure 6.4 is a result of a stimulus that follows a similar pattern outlined by Table 3.5, but with N = 4 and n = 4. The definitions of the variables used in the wave form are the same as the ones used in the Binary protocol case, except for the number of states. Here, the states are only S0, S1, and S2.

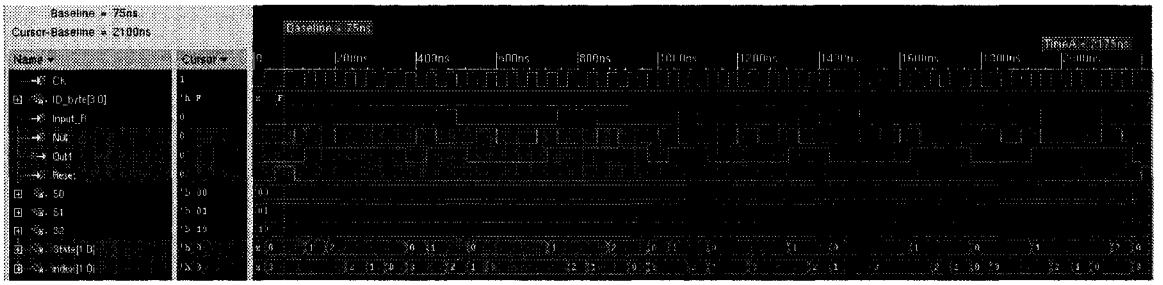

**Figure 6.4: Simulation output for Tag# 15 (N = 4, n = 4): Query-Tree Protocol.**

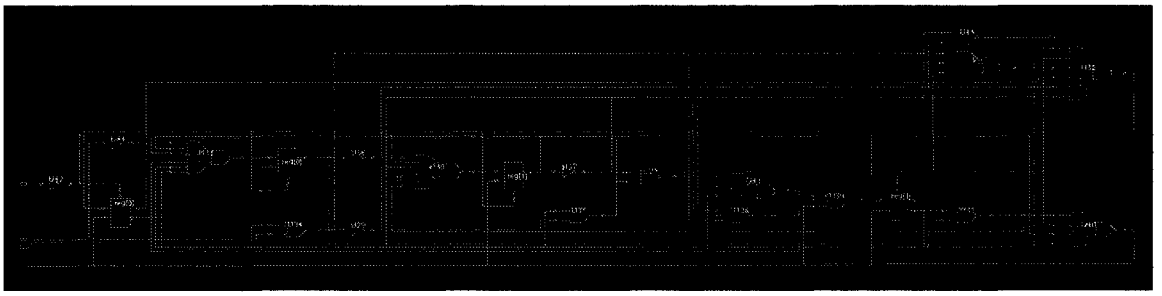

**Figure 6.5: Schematic for Tag# 15 (N = 4): Query-Tree Protocol**

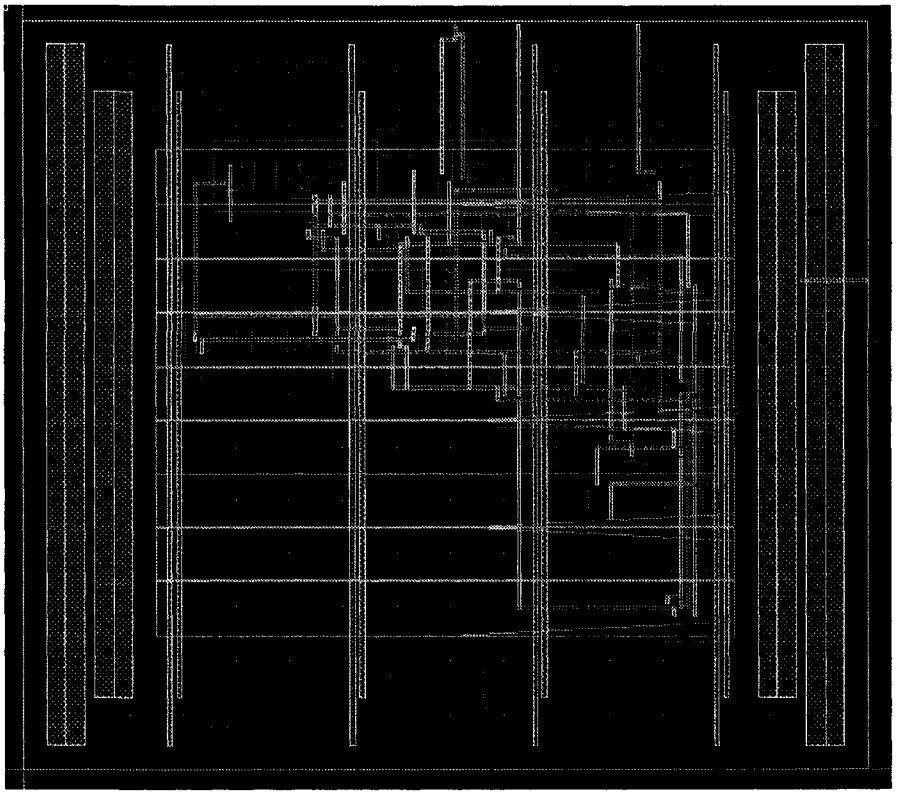

**Figure 6.6: Layout for Tag# 15 (N = 4): Query-Tree Protocol**

## 6.3. Implementation and Simulation of the Improved Query-Tree protocol

Similarly, for Tag# 15 implementations at the different levels of abstraction using this protocol were performed and their corresponding results are displayed in Figure 6.7, 19 and 20. Similar to the RTL Verilog code of the Query Protocol, the RTL Verilog code for this protocol also assumes the condition for transition from state "S0" to "S1" and from "S1" to "S2" to be when "Null" = "1" and "Input_R" = "0". Appendix A.3.1 shows the full code.

Figure 6.7 shows the verification of the functionality of the designed Verilog code under the worst-case scenario. Figure 6.8 shows the schematic that was obtained after

synthesizing the Verilog code using design_vision and Figure 6.9 shows the final layout from Virtuoso.

From Figure 6.7, that show the simulation result for the Verilog code using SimVision, we can see that it takes 2100ns for Tag# 15 to be read, as a result, the total number of clock cycles = 2050ns * 20MHz = 41.

The waveform shown in Figure 6.7 is a result of a stimulus that follows a similar pattern as outlined by Table 3.7, but with N = 4 and n = 4. The definitions of the variables used in the wave form are the same as the ones used in the Binary protocol case, however like the Query protocol, the number of states are only S0, S1, and S2.

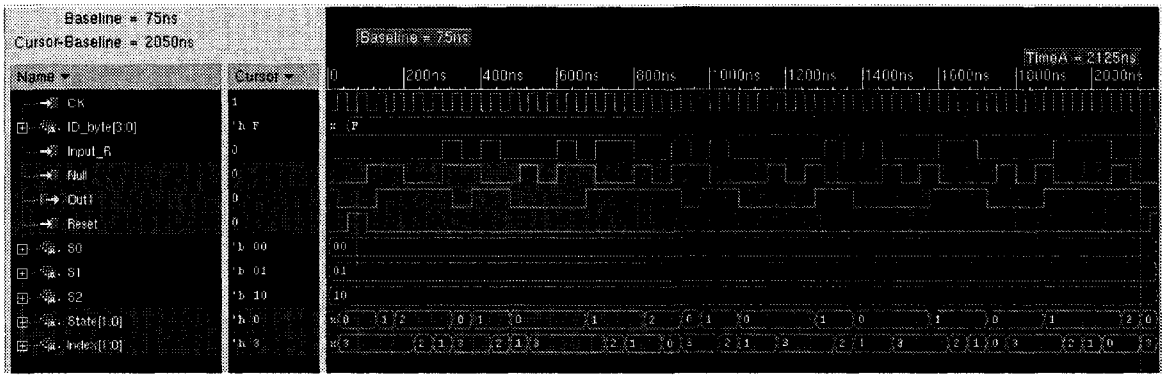


**Figure 6.7: Simulation output for Tag# 15 (N = 4, n = 4): Improved-Query-Tree Protocol.**

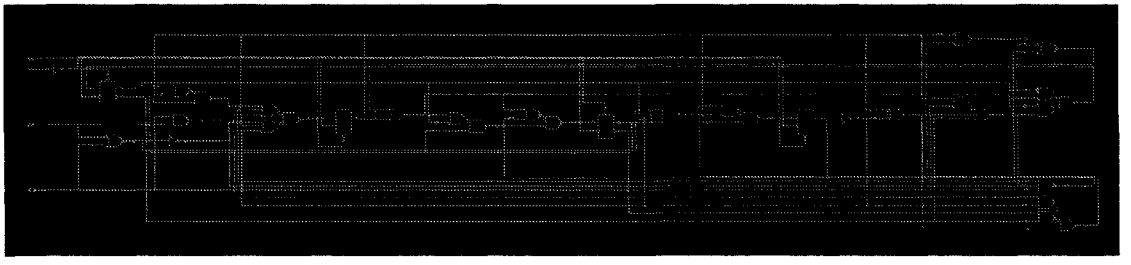

**Figure 6.8: Schematic for Tag# 15 (N = 4): Improved-Query-Tree Protocol**

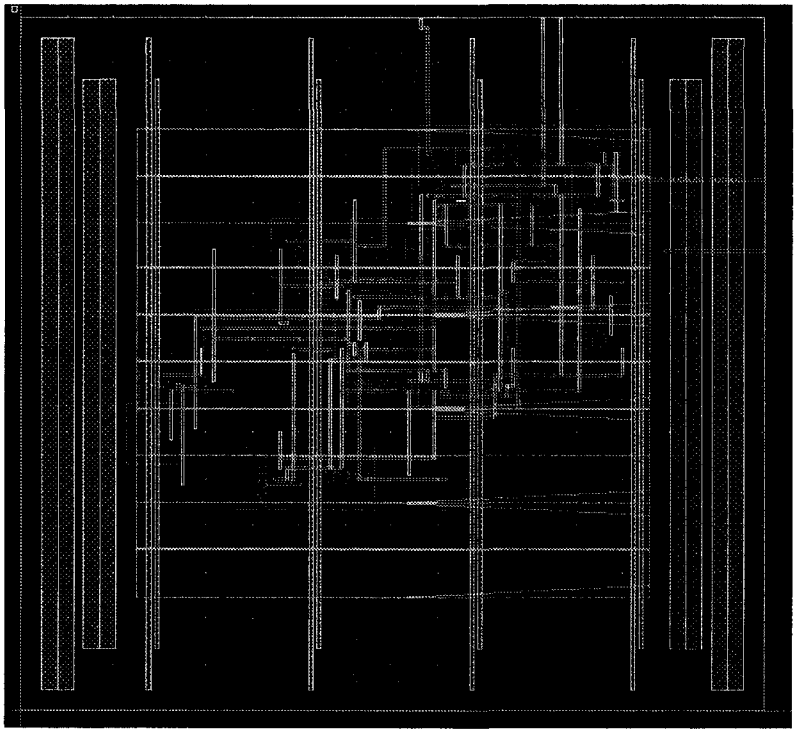

**Figure 6.9: Layout for Tag# 15 (N = 4): Improved-Query-Tree Protocol**

## 6.4. Implementation and Simulation of the Combined-Binary-Tree Protocol

Implementations methods used in the previous three protocols are repeated for this proposed protocol and the corresponding results are shown in Figure 6.10, 6.11 and 6.12. Figure 6.10 shows the simulation output of the top-level design for this protocol. Figure 6.11 shows the schematic that was obtained after synthesizing the Verilog code using design_vision and Figure 6.12 shows the final layout from Virtuoso.

From Figure 6.10, that show the simulation result for the Verilog code, we can see that it takes 1350ns for Tag# 15 to be read, as a result, the total number of clock cycles = 1350ns * 20MHz = 27, way less than the other protocols used in this work.

The waveform shown in Figure 6.10 is a result of a stimulus that follows a similar pattern as outlined by Table 4.1, but with N = 4 and n = 4. Again, the definitions of the variables used in the wave form are the same as the ones used in the Binary protocol case.

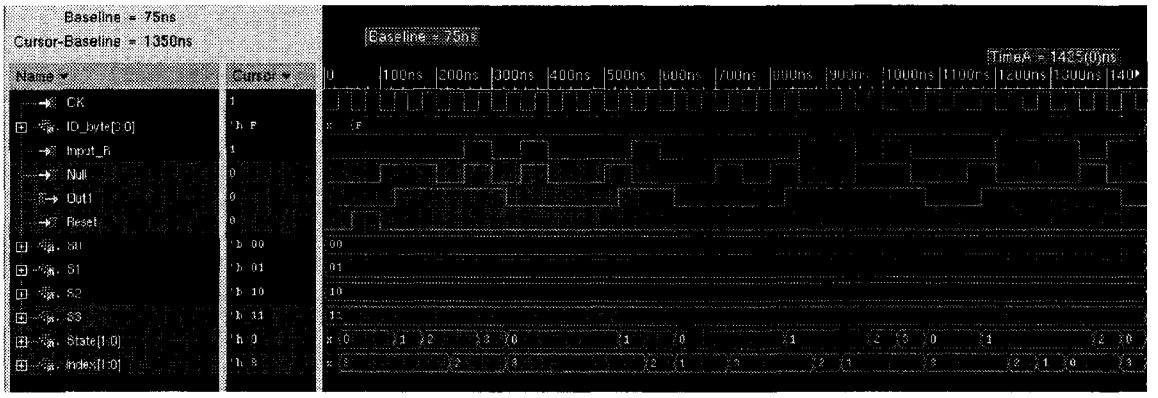


**Figure 6.10: Simulation output for Tag# 15 (N = 4, n = 4): Combined-Query-Tree Protocol.**

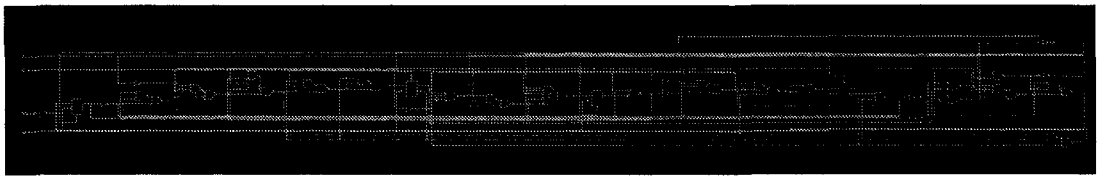

**Figure 6.11: Schematic for Tag# 15 (N = 4): Combined -Query-Tree Protocol**

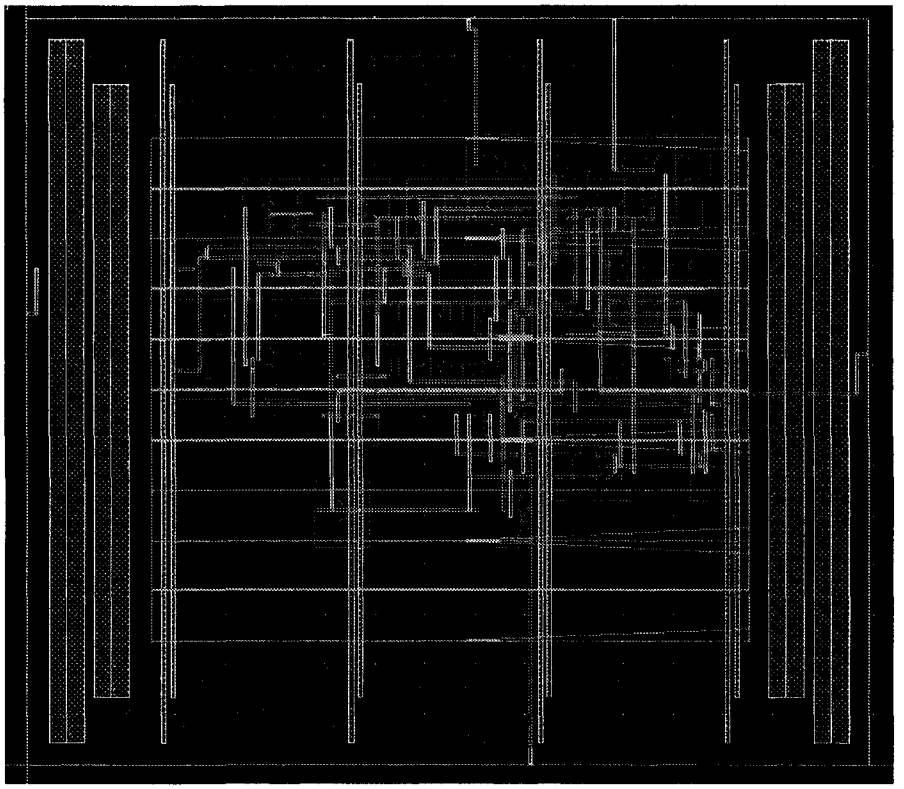

**Figure 6.12: Layout for Tag# 15 (N = 4): Combined -Query-Tree Protocol**

## 6.5. Layout Level Comparison of the four protocols

During layout implementation, the four major protocols discussed in this work were evaluated a) in terms of power consumption and total number of cells and pins used by each protocol generated from "Encounter" and b) in terms of instantaneous power consumption generated in "dfII" environment.

### 6.5.1. Comparison of the four protocols using "Encounter"

For each protocol, the net list file that was generated using "Encounter" ("Encounter" is a tool that we used in our work to place and run our design) was simulated by the respective test bench to generate a value change dump (VCD) file. This VCD file

contains the switching activity for the particular protocol. Then, accurate power consumption was estimated using the VCD file, the capacitance information obtained from "Encounter", and the frequency of operation. We used a frequency of 20MHz in our design.

Power consumption can be accurately estimated using the following equation [13]

$$P_{sw} = 0.5 * C_{load} * V_{dd}^{\ 2} * f_{clk} * E(transitions)$$
$$where, \qquad C_{load} = \sum_{allfanout} C_G^{\ i} + C_{wire} \tag{7.1}$$

And

$$P_{\text{int}} = 0.5 * V_{dd}^{\ 2} * f_{clk} * \sum_{i=1}^{N} \left[ C_G^{\ i} * E^i(transitions) \right] \tag{7.2}$$

Where,

$P_{sw}$ and $P_{int}$ are switching and internal powers; $N_{in}$ is # of internal nodes

$C_G^{\ i}$ is gate capacitance of $i^{th}$ fan-out and $C_{wire}$ is interconnect capacitance of the driver net

$E^i$(transitions) is expected # of transitions at node i.

$V_{dd}$ is the supply voltage ($V_{dd}$ = 1.62V in our work)

The capacitance information was obtained from "Encounter" after extracting RC. Table 6.2 shows the capacitance information for the Binary-Tree Protocol design.

From Table 6.2, the total capacitance ($C_{load}$) is calculated to be 0.99641pF. For other protocols, information about their capacitance was obtained using similar methods, but they are left out here to prevent repetition.

Table 6.3 shows summery of the comparative metric results for the four protocols for N = 4 and n = 4.

In Table 6.3, we see that for the protocols under study, the total clock cycles obtained from simulation results exactly matches the predicted clock cycles from protocol level analysis. Other metrics, such as power and energy do not reflect the results obtained in protocol level analysis. The main reason for the discrepancies is due to the fact that power analysis in protocol level was confined to state transitions only, but in layout level it was highly dependant on total capacitance and total switching activity.

As can be seen from Table 6.3, as a result of increase in total capacitance and total switching activity going from Binary to Query to Improved-Query to Combined Binary-Query trees, their corresponding power increases in that order. This clearly is different from the order of power increase we found in protocol level analysis where the order of increase was from Combined to Query, Improved-Query to Binary. We should note that the total power consumption shown in Table 6.3 is not based on equal clock cycles. If we seek total power consumption under equal cycles, say selecting $Clk_B$ to be the reference clock cycle, then Total power (uW) of Query, Improved-Query, and Combined will be 49.55, 50.54, and 33.64 respectively. Hence the Combined Binary-Tree will provide the lowest power consumption.

From energy point of view (where Total Energy = Total Power * Total number of Clock cycles * Period), the Combined Binary-Tree outperforms the other tree protocols.

**Table 6.2: Capacitance information for the Binary-Tree Protocol**

| Binary Protocol (N = 4, n = 4) | | | | | | | |
|---|---|---|---|---|---|---|---|
| The wireCap, pinCap and totalCap are in the unit of pF and | | | | | | | |
| The netLength is in the unit micron. | | | | | | | |
| The capPerUnitLen is in pF per micron. | | | | | | | |
| | | | | | | | |
| netName | wireCap | pinCap | totalCap | Net Length | wireCapPer UnitLen | nr Fanout | |
| CK | 0.002197 | 0.010417 | 0.012614 | 16.36 | 1.34E-04 | 1 | 0.012614 |
| Null | 0.004467 | 0.004936 | 0.009403 | 33.19 | 1.35E-04 | 1 | 0.009403 |
| Input_R | 0.00186 | 0.003457 | 0.005317 | 13.82 | 1.35E-04 | 1 | 0.005317 |
| Reset | 0.00186 | 0.003457 | 0.005317 | 13.82 | 1.35E-04 | 1 | 0.005317 |
| Out1 | 0.002233 | 0 | 0.002233 | 16.59 | 1.35E-04 | 1 | 0.002233 |
| CK__L2_N0 | 0.006767 | 0.010246 | 0.017013 | 50.28 | 1.35E-04 | 4 | 0.068052 |
| CK__L1_N0 | 0.00224 | 0.010417 | 0.012657 | 16.64 | 1.35E-04 | 1 | 0.012657 |
| N30 | 0.001488 | 0.00184 | 0.003328 | 11.04 | 1.35E-04 | 1 | 0.003328 |
| N31 | 0.002132 | 0.00184 | 0.003972 | 15.84 | 1.35E-04 | 1 | 0.003972 |
| n2 | 0.001061 | 0.004032 | 0.005093 | 7.88 | 1.35E-04 | 1 | 0.005093 |
| n3 | 0.000464 | 0.003746 | 0.00421 | 3.46 | 1.34E-04 | 1 | 0.00421 |
| n4 | 0.009321 | 0.010072 | 0.019393 | 69.4 | 1.34E-04 | 3 | 0.058179 |
| n5 | 0.008933 | 0.01629 | 0.025223 | 66.855 | 1.34E-04 | 4 | 0.100892 |
| n6 | 0.009894 | 0.023573 | 0.033467 | 73.5 | 1.35E-04 | 6 | 0.200802 |
| n7 | 0.005579 | 0.014191 | 0.01977 | 41.98 | 1.33E-04 | 3 | 0.05931 |
| n8 | 0.001997 | 0.00348 | 0.005477 | 14.84 | 1.35E-04 | 1 | 0.005477 |
| n9 | 0.001192 | 0.007355 | 0.008547 | 8.9 | 1.34E-04 | 2 | 0.017094 |
| n10 | 0.007085 | 0.012581 | 0.019666 | 52.72 | 1.34E-04 | 3 | 0.058998 |
| n12 | 0.001135 | 0.003365 | 0.0045 | 8.44 | 1.34E-04 | 1 | 0.0045 |
| n13 | 0.00509 | 0.007795 | 0.012885 | 38.06 | 1.34E-04 | 2 | 0.02577 |
| n14 | 0.00175 | 0.004374 | 0.006124 | 13.26 | 1.32E-04 | 1 | 0.006124 |
| n15 | 0.001253 | 0.004743 | 0.005996 | 9.4 | 1.33E-04 | 1 | 0.005996 |
| n16 | 0.003335 | 0.010217 | 0.013552 | 24.76 | 1.35E-04 | 2 | 0.027104 |
| n17 | 0.001621 | 0.003175 | 0.004796 | 12.06 | 1.34E-04 | 1 | 0.004796 |
| n18 | 0.009981 | 0.01782 | 0.027801 | 74.18 | 1.35E-04 | 4 | 0.111204 |
| n19 | 0.000611 | 0.003175 | 0.003786 | 4.62 | 1.32E-04 | 1 | 0.003786 |
| State[1] | 0.00236 | 0.011706 | 0.014066 | 17.54 | 1.35E-04 | 3 | 0.042198 |
| State[0] | 0.006517 | 0.019926 | 0.026443 | 48.5 | 1.34E-04 | 5 | 0.132215 |

**Table 6.3: Implementation results for the Binary, Query, Improved-Query, and Combined-Binary-Query protocols for N = 4 and n = 4**

| Metrics | Binary | Query | Improved-Query | Combined |
|---|---|---|---|---|
| Clock cycles (Clk) | 31 | 42 | 41 | 27 |
| Total capacitance (pF) | 0.99641 | 1.165214 | 1.317833 | 1.49723 |
| Switching power (uW) | 4.1094 | 4.485 | 5.1409 | 5.9422 |
| Internal power (uW) | 31.593 | 32.036 | 33.044 | 32.653 |
| Leakage power (uW) | 0.028176 | 0.028139 | 0.031744 | 0.029359 |
| Total power (uW) | 35.731 | 36.576 | 38.217 | 38.625 |
| Total activity | 574 | 784 | 873 | 757 |
| Total Energy (pJ) | 55.38305 | 76.809 | 78.345 | 52.144 |
| Physical parameters | | | | |
| # of cells | 23 | 24 | 25 | 34 |
| # of nets | 28 | 29 | 31 | 39 |
| # of pins | 85 | 93 | 99 | 123 |
| # of I/Os | 5 | 5 | 5 | 5 |

## 6.5.2. Comparison of the four protocols in "dfII"

After the physical implementation of the four protocols was performed in "dfII" (dfII is a Cadence Design FrameworkII directory where physical implementation and final design verification is done), physical testing circuit was setup to measure the instantaneous power as well as to verify the functionality of each protocols. In this testing, all inputs are digital where "1" represents 1.8V and "0" represents 0V

Figure 6.13 shows the circuit used to test the Binary-Tree Protocol for N = 4 and n = 2. In this circuit, a square-wave with a period of 50ns is used as a clock input. "Reset" is set to "1" at the first positive edge clock to start the reading process and reset to "0" for the rest of reading cycle. After the application of "Reset", the values of "Null" and "Input_R" are set to follow the pattern outlined in Table 3.1 (Identification Process of Two tags (n = 2) by Binary-Tree Protocol for N = 4).

Simulation is done for 925ns and the results of the simulation including the instantaneous power consumption are plotted. We used 925ns simulation time for this protocol because: the total clock cycles obtained either from Table 3.1 or equation (3.2) (for N = 4, n = 2) is 17, and after including the initial one clock cycle for "Reset" and another half clock cycle for final state return from "S3" to "S0"; the total clock cycles will be 18.5. Hence, 18.5 * 50ns = 925ns.

For the rest of the protocols, the same testing setup as Figure 6.13 is used. The exception for each protocol include: the values of "Null" and "Input_R", the device under test, and the simulation time.

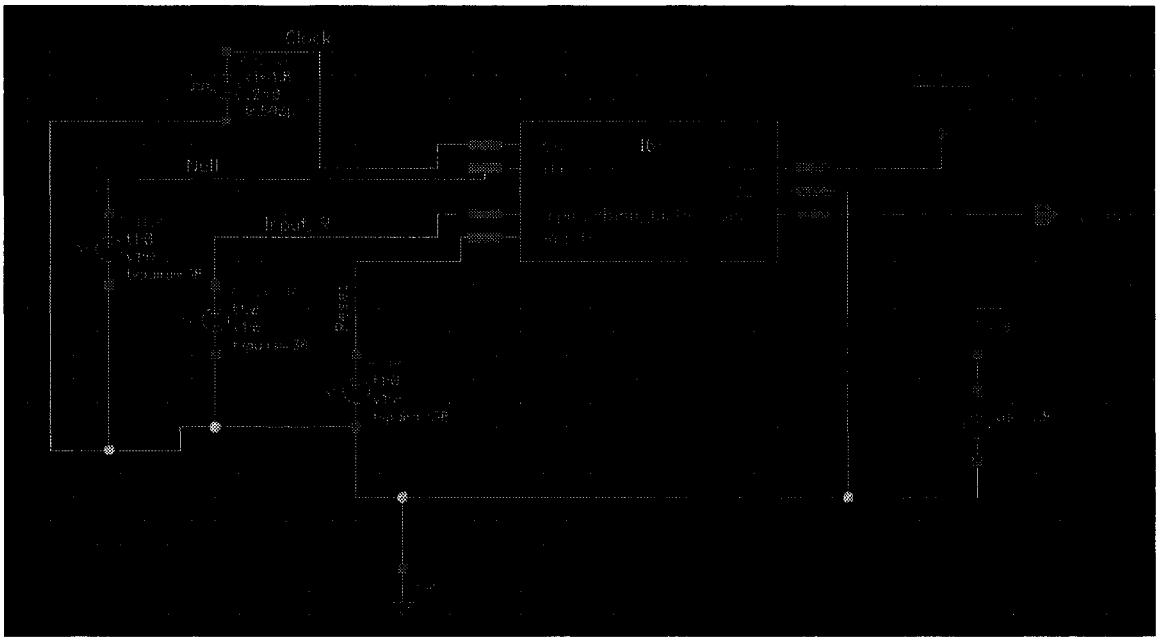

**Figure 6.13: Test setup to determine instantaneous power by Binary-Tree Protocol (N = 4, n = 2)**

### 6.5.2.1. Results from Physical Implementation for Binary-Tree Protocol

Figure 6.14 shows the results of the simulation for Binary-Tree after simulation for 925ns. The figure displays "Supply current", current through $V_{dd}$ and "Output" for input vectors of "Reset", "Input_R", and "Null". From the figure, we verified that the result (output) we are getting accurately follows Table 3.1.

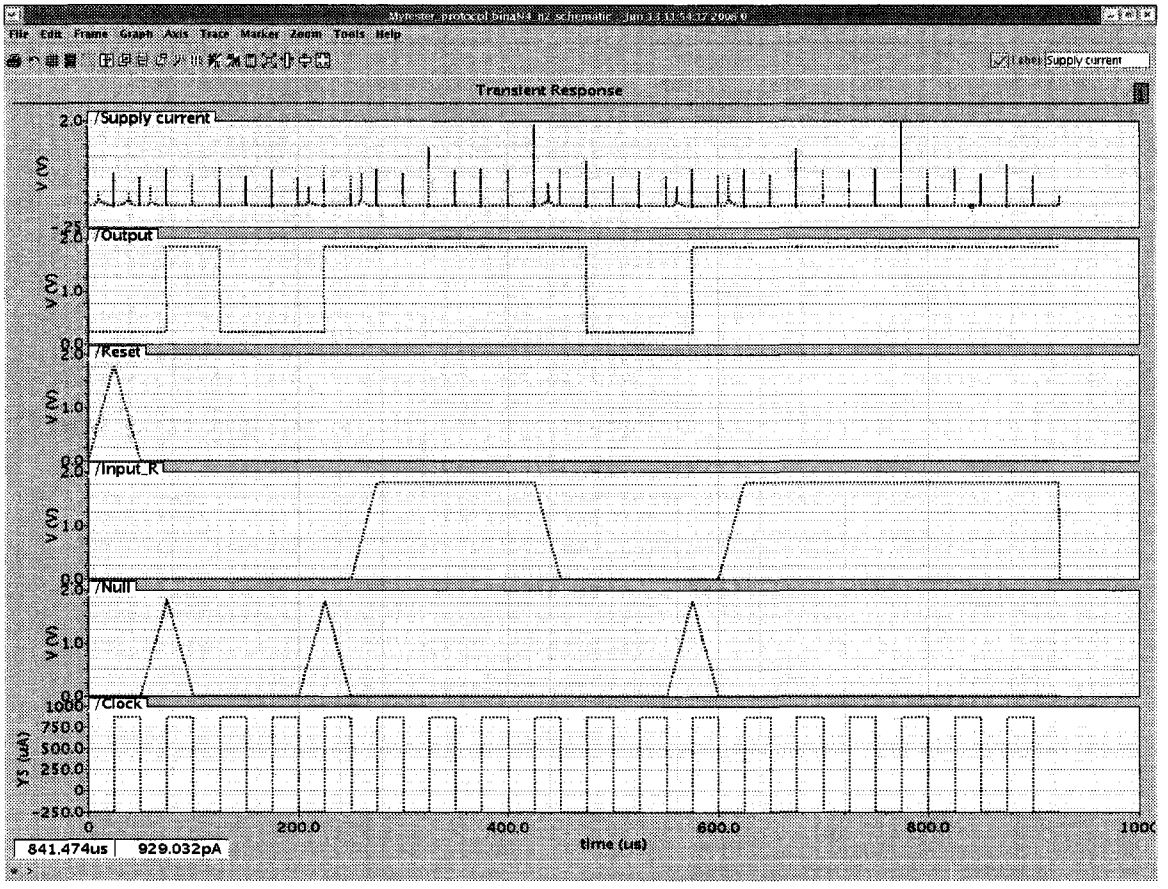


**Figure 6.14: Results of Simulation for Binary-Tree Protocol (N = 4, n = 2)**

Figure 6.15 shows the instantaneous power consumption of this protocol for the simulation period. The average power is also displayed on the figure.

The instantaneous power is shown to have a maximum value of 1.763uW at 775us. The average power is calculated to be 6.572uW.

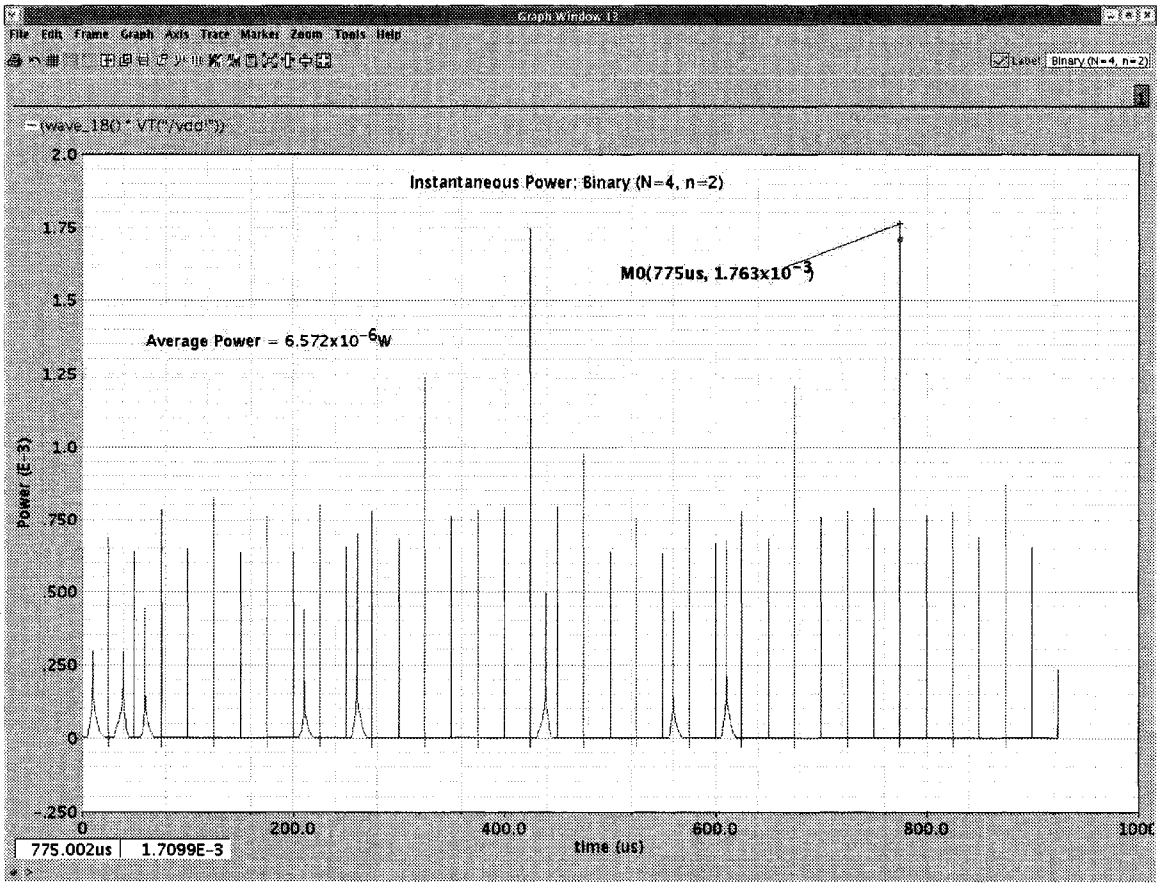


**Figure 6.15: Instantaneous power for Binary-Tree Protocol (N = 4, n = 2)**

### 6.5.2.2. Results from Physical Implementation for Query-Tree Protocol

As mentioned in section 6.5.2, the testing setup is similar to Figure 6.13 but the assignment of "Null" and "Input_R" need to follow a modified Table 3.5. Table 3.5 is used for N = 5, n = 2; hence similar table can be constructed for N = 4, n = 2.

Simulation is done for 975ns and the results of the simulation including the instantaneous power consumption are plotted. We used 975ns simulation time for this protocol because: the total clock cycles obtained using equation (3.8) (for N = 4, n = 2) is 18, and after including the initial one clock cycle for "Reset" another half clock cycle for final state return from "S2" to "S0"; the total clock cycles will be 19.5. Hence, 19.5 * 50ns = 975ns.

Figure 6.16 shows the results of the simulation for Query-Tree after simulation for 975ns. The figure displays "Supply current" and "Output" for input vectors of "Reset", "Input_R", and "Null". From the figure, we verified that the result (output) we are getting accurately describes the total reading of Query-Tree Protocol for N = 4, n = 2.

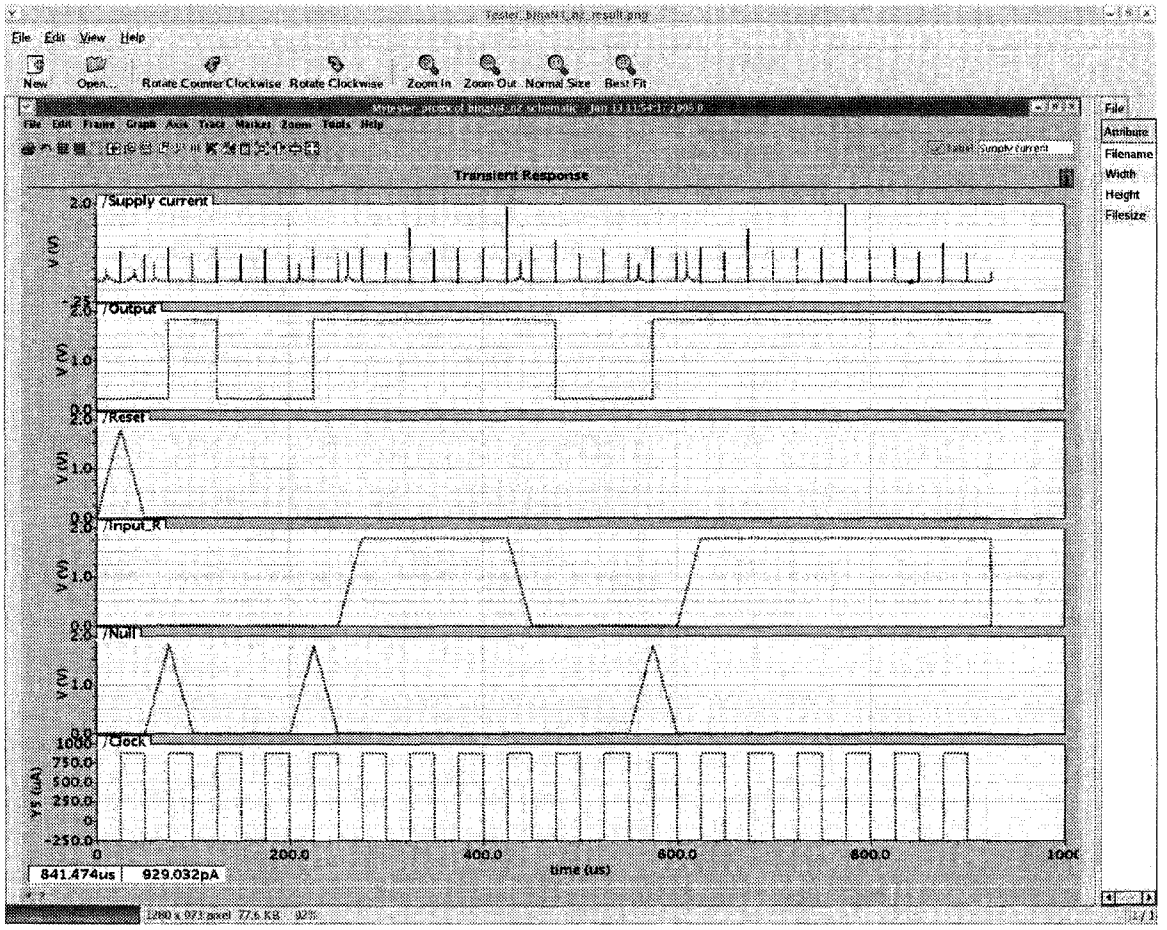


**Figure 6.16 Results of Simulation for Query-Tree Protocol (N = 4, n = 2)**

Figure 6.17 shows the instantaneous power consumption of this protocol for the simulation period. The average power is also displayed on the figure.

The instantaneous power is shown to have a maximum value of 1.577uW at 325us. The average power is calculated to be 8.375uW.

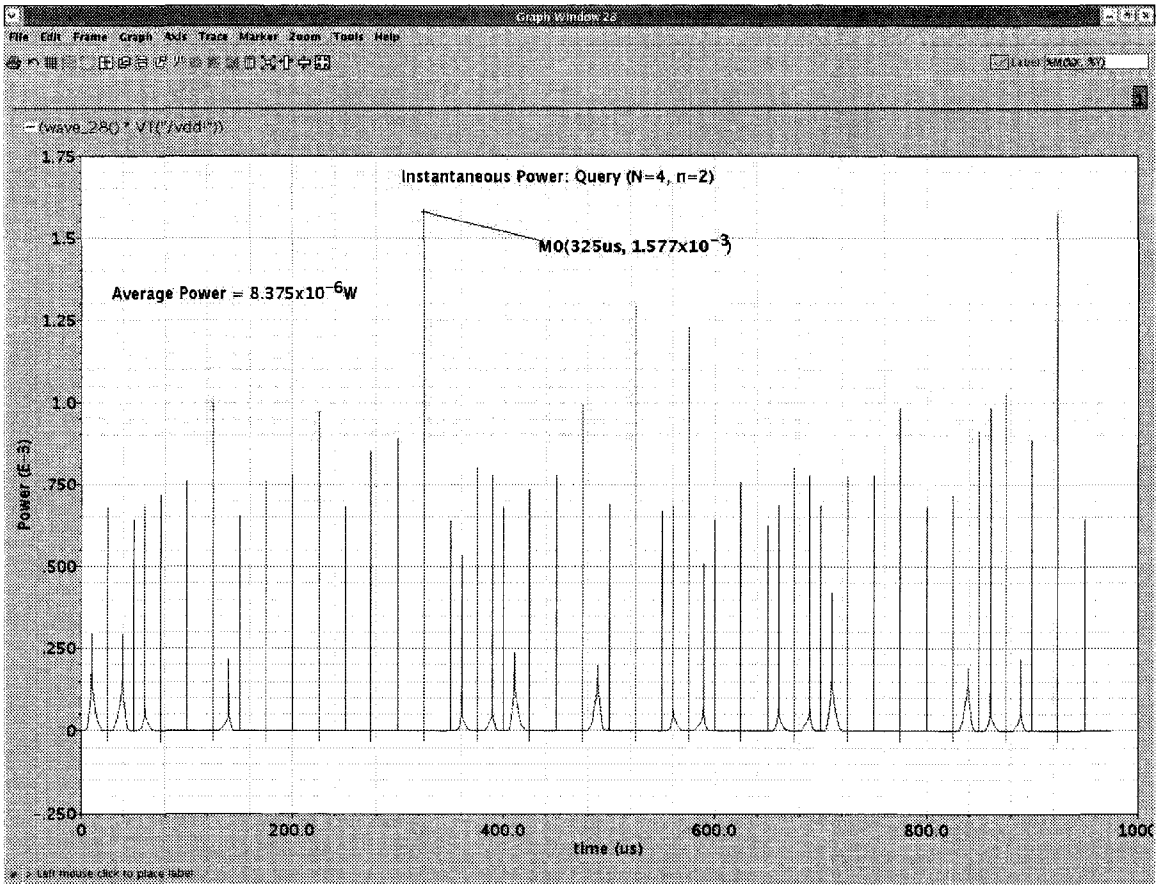


**Figure 6.17: Instantaneous power for Query-Tree Protocol (N = 4, n = 2)**

### 6.5.2.3. Results from Physical Implementation for Improved Query-Tree Protocol

The testing setup is also similar to Figure 6.13; but the assignment of testing vectors of "Null" and "Input_R" need to follow a modified Table 3.7. Table 3.7 is constructed for N = 5, n = 2; hence similar table can easily be constructed for N = 4, n = 2.

Simulation is done for 975ns and the results of the simulation including the instantaneous power consumption are plotted. We used 975ns simulation time for this protocol because:

the total clock cycles obtained using equation (3.18) (for N = 4, n = 2) is 18, and after including the initial one clock cycle for "Reset" another half clock cycle for final state return from "S2" to "S0"; the total clock cycles will be 19.5. Hence, 19.5 * 50ns = 975ns.

Figure 6.18 shows the results of the simulation for Improved Query-Tree after simulation for 975ns. The figure displays "Supply current" and "Output" for input vectors of "Reset", "Input_R", and "Null". From the figure, we verified that the result we are getting accurately describes the total reading of the Improved Query-Tree Protocol for N = 4, n = 2.

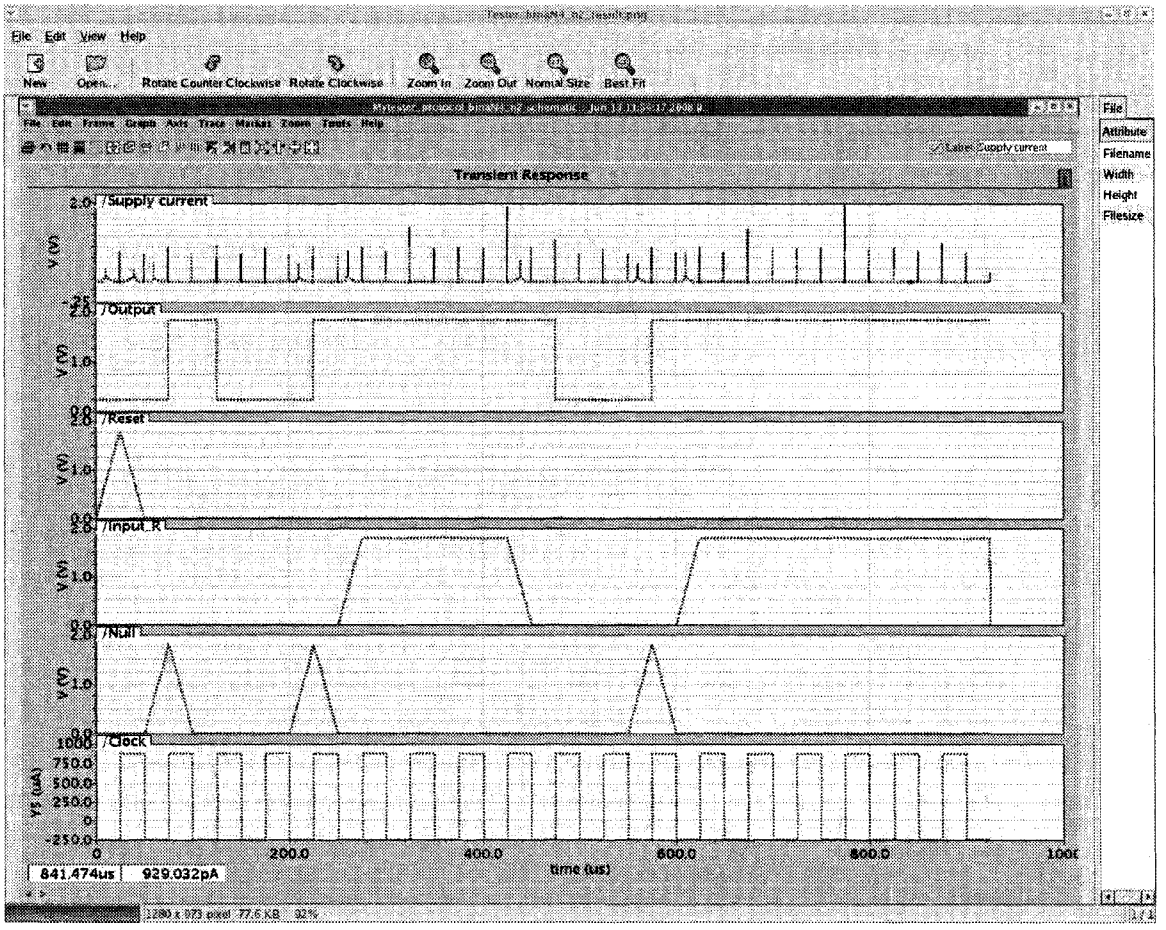


**Figure 6.18: Results of Simulation for Improved Query-Tree Protocol (N = 4, n = 2)**

Figure 6.19 shows the instantaneous power consumption of this protocol for the simulation period. The average power is also displayed on the figure.

The instantaneous power is shown to have a maximum value of 1.865uW at 525us. The average power is calculated to be 10.34uW.

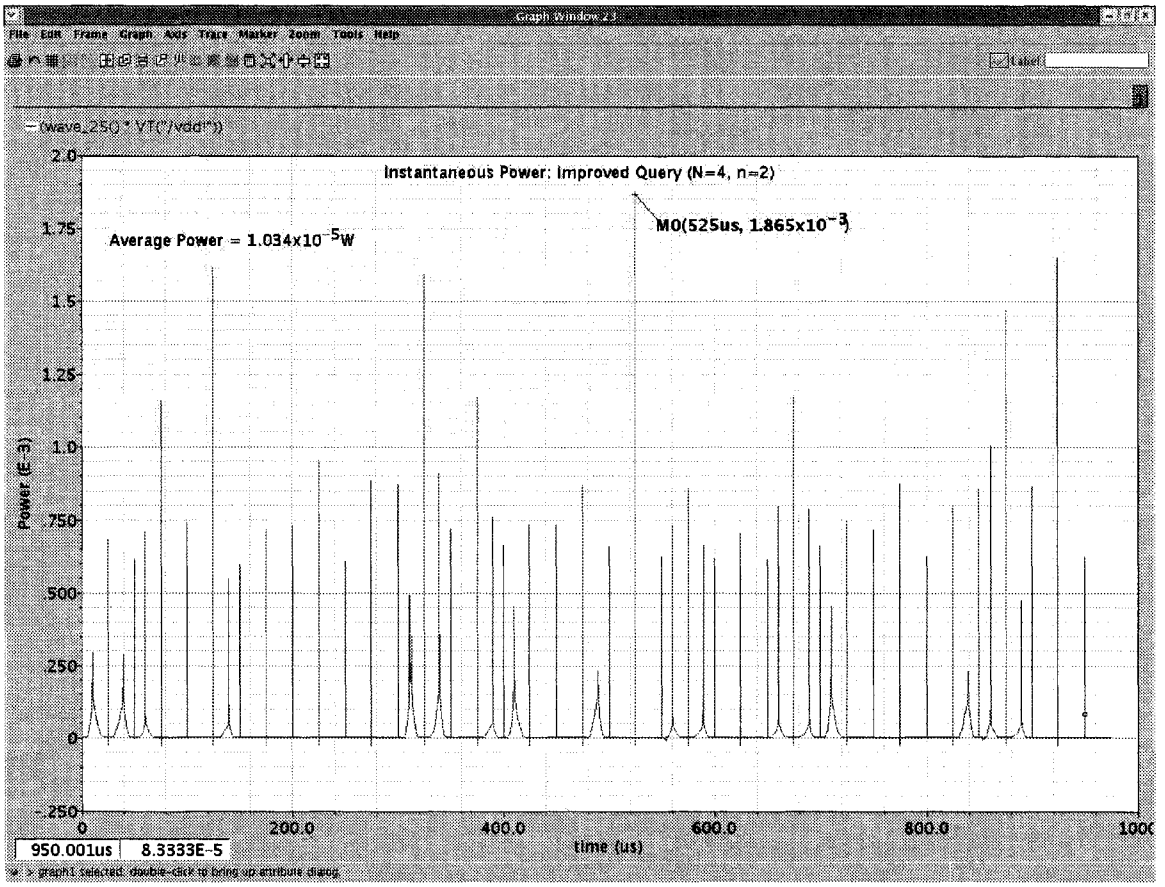


**Figure 6.19: Instantaneous power for Improved Query-Tree Protocol (N = 4, n = 2)**

### 6.5.2.4. Results from Physical Implementation for Combined-Binary-Tree Protocol

The testing setup is again similar to Figure 6.13; but the assignment of "Null" and "Input_R" need to follow a modified Table 4.1. Again, Table 4.1 is constructed for N = 5, n = 2; hence similar table can easily be constructed for N = 4, n = 2.

Simulation is done for 725ns and the results of the simulation including the instantaneous power consumption are plotted. We used 725ns simulation time for this protocol because: the total clock cycles obtained using equation (4.2) (for N = 4, n = 2) is 13, and after including the initial one clock cycle for "Reset" another half clock cycle for final state return from "S3" to "S0"; the total clock cycles will be 14.5. Hence, 14.5 * 50ns = 725ns.

Figure 6.20 shows the results of the simulation for Combined-Binary-Query-Tree after simulation for 725ns. The figure displays "Supply current" and "Output" for input vectors of "Reset", "Input_R", and "Null". From the figure, we verified that the result we are getting accurately describes the total reading of the Combined-Binary-Query-Tree Protocol for N = 4, n = 2.

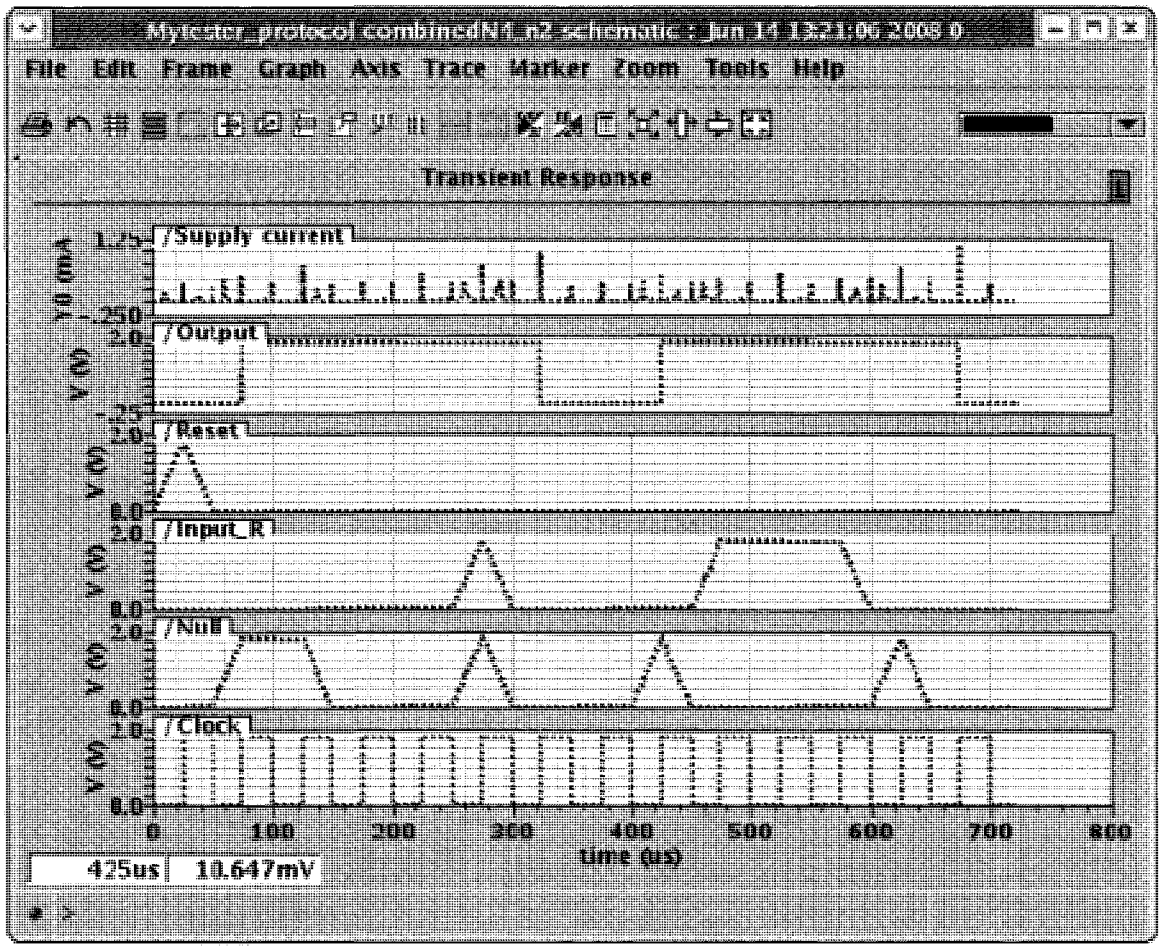


**Figure 6.20: Results of Simulation for Combined-Binary-Query-Tree Protocol (N = 4, n = 2)**

Figure 6.21 shows the instantaneous power consumption of this protocol for the simulation period. The average power is also displayed on the figure.

The instantaneous power is shown to have a maximum value of 2.005uW at 675us. The average power is calculated to be 23.3uW.

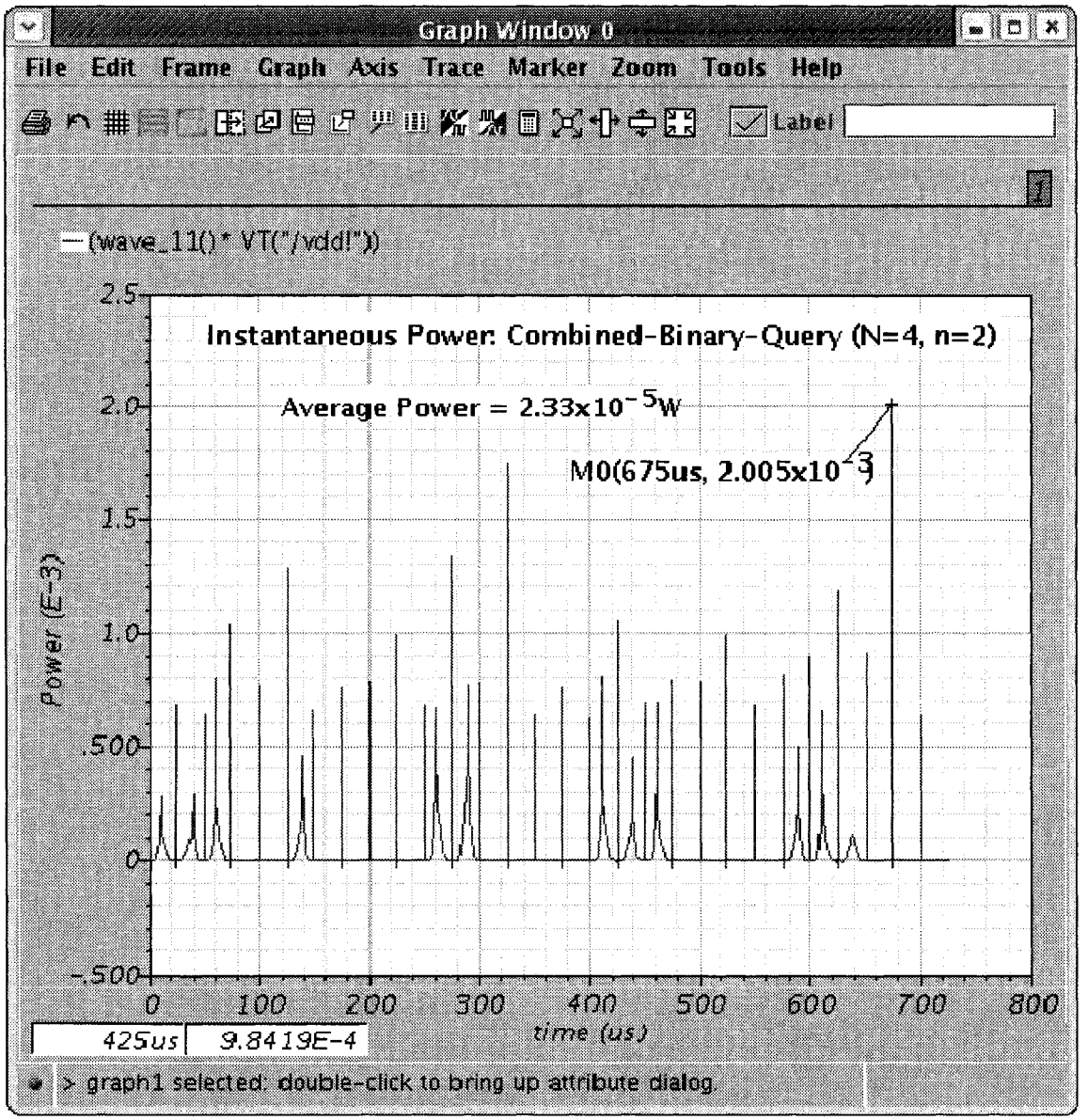


**Figure 6.21: Instantaneous power for Combined-Binary-Query-Tree Protocol (N = 4, n = 2)**

Note that when equal reading time is taken for comparison, the average power for this protocol (Proposed protocol) will reduce by 725/925 = 0.784; where 925ns is the reading time for binary tree protocol and 725ns for proposed protocol. Hence, when equal reading time of 925ns is considered, the average power consumed for binary, query, improved-query, and combined-binary-query protocols will be 6.572uW, 8.828uW, 10.899uW, and 18.262uW respectively.

# Chapter 7 Conclusions and Future Work

## 7.1. Conclusions and Contributions

We have studied protocol-level and implementation-level performance analysis for anti-collision protocols in RFID systems. Since the performance of a particular protocol depends on the ID distribution of the tags, our discussions are based on the worst-case scenario for a given number of tags available. In protocol level, for binary-tree, query-tree and improved query-tree, formulas for the number of state transitions and clock cycles were derived. In particular, we have proposed an improved protocol that combines the binary-tree and query-tree protocols and derived the number of state transitions and clock cycles for this protocol as well. The various protocols were evaluated in terms of number of transitions, clock cycles, power consumptions, and energy. In implementation-level, we designed the various protocols in different levels of abstractions and performed accurate power analysis in layout level. From the power analysis obtained in layout level, where we used small value of ID bits, N = 4, we concluded that power consumption was dominated by the total capacitance and total switching activity not by the sole consideration of state transitions as in the case for protocol-level analysis. But for implementation that uses large values of N, the state switching activities will be dominant over the capacitances, and the corresponding layout-level power consumption is expected to be consistent with the results obtained through protocol-level analysis. We also asserted that, the proposed protocol not only needs less clock cycles compared to other protocols, but also outperforms them in terms of power as well as energy consumptions.

## 7.2. Future Work

An RFID tag, in addition to anti-collision protocol block, is mainly consisted of power generator circuit (AC/DC converter), modulator, decoder, encoder, power control, instruction sequencer, and memory blocks [3, 14]. Our study is concentrated only on analyzing and realizing of different anti-collision protocols; hence there are many blocks in RFID tag that need research. The current antenna and capacitor combination for power receiving and AC/DC conversion needs further study. The analog circuitry involving modulator, decoder, and encoder also requires further study.

In our study, the protocol-level and implementation-level performance analysis was based on purely digital inputs. When we develop the state diagrams (Figure 2.2, 2.3, and 4.1) that represent the respective protocols, we assumed that the conditions (inputs) for state transitions were purely digital ones. The fact is we are dealing with wireless communication that introduces error during receiving and transmission of signals between reader and tags. As a result, we need to consider the effect of error in the analysis. The future work need to consider introduction of error when developing state diagrams for various protocols.

# References

[1] F. Zhou and C. Chen, et al, "Evaluating and Optimizing Power Consumption of Anti-Collision Protocols for Applications in RFID Systems," in *Proc. of 2004 IEEE International Symposium on Low Power Electronics and Design*, pp. 357-362, 2004.

[2] MICROCHIP, Appl. Note AN680, pp. 1-7, 1998.

[3] K. Finkenzeller, *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*, 2nd ed. Wiley, 2003.

[4] C. Law, K. Lee, K. Y. Siu, "Efficient Memoryless Protocol for Tag Identification," In Proc. of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, pp. 75–84, 2000.

[5] J. I. Capetanakis, "Tree Algorithms for Packet Broadcast Channels," *IEEE Trans. Information Theory*, vol. 25, pp. 505-515, Sept. 1979.

[6] D. R. Hush and C. Wood, "Analysis of Tree Algorithms for RFID Arbitration," in *Proc. of IEEE International Symposium on Information Theory*, pp. 107-112, 1998.

[7] S. Lee, S.D. Joo, and C.W. Lee, "An Enhanced Dynamic Framed Slotted Aloha Algorithm for RFID Tag Identification," in *Proc. of Mobiquitous 2005*, pp. 166-172.

[8] J. Myung and W. Lee, "Adaptive Binary Splitting: A RFID Tag Collision Arbitration Protocol for Tag Identification," *IEEE Communications Letter*, vol. 10, no. 3, pp. 141-146, March 2006.

[9] Shih, D., Sun, P., Yen, D., Huang, S., "Taxonomy and survey of RFID anti-collision protocols", *Elsevier Computer Communications*, vol. 29, pp. 2150–2166, 2006

[10] Product version 3.4, *Verilog –XL Reference*, Cadence, Jan. 2002

[11] Man Page for design_vision, snpsDocBrouser::getManIndex 2 commands

[12] Product Version 5.2.5, *Encounter Menu Reference*, Cadence

[13] F. Zhou, D. Jing, C. Juang and H. Min, "Optimizing the Power Consumption of Passive Electronic Tags For Anti-collision Schemes," *In Proceedings of the 5th ASICON*, Beijing, China, pp. 1213–1217, 2003.

[14] S. Lewis "A BASIC INTRODUCTION TO RFID TECHNOLOGY AND ITS USE IN THE SUPPLY CHAIN", White Paper, Jan. 2004

[15] V. Namboodiri, L. Gao, "Energy-Aware Tag Anti-Collision Protocols for RFID Systems", *in Proceedings of IEEE PerCom*, White Plains, NY, March 2007.

[16] J. Myung and W. Lee, "An Adaptive Memoryless Tag Anti-Collision Protocol for RFID Networks," *The 23rd Conference of the IEEE Communications Society*, 2005.

[17] Document ICI-096, *Tutorial on CMC's Digital IC Design Flow*, Canadian Microelectronics Corporation, Oct. 2002.

# Appendix A RTL Codes and Test benches

For every protocol used in our work, an RTL code (Verilog code) was written and its functionality was verified using a respective test bench (written in Verilog). In this section, the top-level module (Verilog code) as well the corresponding test bench for Binary-Tree Protocol, Query-Tree Protocol, Improved-Query-Tree Protocol, and Combined-Binary Tree Protocol are listed.

The following file names are given to the protocols:

- For Binary-Tree Protocol: bina_last4
- For Query-Tree Protocol: query_last4
- For Improved-Query-Tree Protocol: impquery_last4
- For Combined-Binary-Tree Protocol: combined_last4

In the file names, "last" is used to represent the last tag (or worst-case tag) and "4" is used to indicate a 4-bit ID.

# A.1. RTL Code and Test bench for the Binary-Tree Protocol

## A.1.1. RTL Code for the Binary-Tree Protocol

```
/*
Verilog Top-level module for Binary Protocol includes the sequential, combinational and ID
memory of the Binary-Tree Protocol for the last tag, tag # 1111.

Out1 is used to determine the status of the state:
Out1 = 1 ---> enquiry bit is same as the ID bit
Out1 = 0 ---> enquiry bit is different from ID bit, hence state is in S0

If tag # had been different from 1111, then additional port Out2 would have been needed such
that:
    Out1 = 1 & Out1 = 0 ---> present ID bit is 1
    Out1 = 0 & Out1 = 1 ---> present ID bit is 0
    Out1 = 0 & Out1 = 0 ---> state is in S0
*/

module bina_last4 (CK,Null,Input_R,Reset,Out1);

parameter S0=2'b00;
parameter S1=2'b01;
parameter S2=2'b11;
parameter S3=2'b10;

input CK;              //System clock from the reader
input Null;            //Start from the reader
input Input_R;         //Received bit from the reader
input Reset;
output Out1;           //Out1 used to output the data
reg Out1;
//Out1 stores the current bit of the ID to compare it with the received bit from the reader
reg [3:0] ID_byte;     //Register to ID of the tag
reg [1:0] State;       //Used for the four states
reg [1:0] index;       //Index to point the current bit in ID
```

```
//Sequential part: Initialization and changing of states
always @ (posedge CK or posedge Reset)
begin
    if (Reset)
    begin
        State       <= S0;
        ID_byte     <= 4'b1111;
        Index       <= 2'b11;
    end
    else
    begin
        case (State)
        S0:
            if (Null)
            begin
                State    <= S1;
                index    <= 2'b11;
            end
            else
            begin
                State    <= S0;
                index    <= 2'b11;
            end
        S1:
            if (Input_R ^ ID_byte[index])
            begin
                State    <= S0;
                index    <= 2'b11;
            end
            else
            begin
                State    <= S2;
                index    <= index;
            end
        S2:
            if (index==1)
            begin
                State    <= S3;
                index    <= index-1;
            end
```

```
            else
            begin
                State    <= S1;
                index    <= index -1;
            end
        S3:
            begin
                State    <= S3;
                index    <= index;
            end
        default:
            begin
                State    <= S0;
                index    <= 2'b11;
            end
        endcase
    end
end

always @ (State)
begin
    if (State==S0)
    begin
        Out1 = 1'b0;
    end
    else
    begin
        Out1 = ID_byte[index];
    end
end
endmodule
```

## A.1.2. Test bench for the Binary-Tree Protocol

```
//Tester for Tag # 1111 using the binary protocol

`timescale 1 ns / 1 ps

module bina_last4_tb;

parameter in1=3, in2=4;
reg CK,Null,Input_R,Reset;
reg [in1-1:0] Input;
wire Out1;
integer index1, index2;

bina_last4 u0(CK,Null,Input_R,Reset,Out1);
initial
    begin
        CK = 0;
        Null = 0;
        Input_R = 0;
        Reset = 0;
        Input = 7'b100;
    end

initial
    begin
        $dumpfile ("bina_last4_encounter.dump");
        $dumpvars (1,u0);
        repeat (1) @(negedge CK); Reset = 1;
        repeat (1) @(negedge CK); Reset = 0;    Null = 1;
    //Use 3 cycles before Null = 0, to implement the No response condition
        repeat (3) @(negedge CK);

        for(index2=in2-1;index2>=0;index2=index2-1)
        begin
            Input_R = Input[in1-1];
            for(index1=in1-2;index1>=0;index1=index1-1)
            begin
                Null = 0;
                repeat (2) @(negedge CK);
```

```
    //    Condition to test for Reset
    /*
                if (index2==2 && index1==1)      Reset = 1;
                else
    */
    //    Condition to test for wrong input
    /*
                if (index2==2 && index1==1)      Input_R = 0;
                else
    */
                Input_R = Input[index1];
            end
            repeat (2) @(negedge CK);
            Input = Input+1;
            Null = 1;
            repeat (1) @(negedge CK);
        end
        $dumpoff;
        Null = 0;
        Reset = 1;     //To show changing from S3 to S0
        #20 $finish;
    end

always
    #25 CK = ~CK ;

//To display inputs and output of the design in table form
initial
    begin
        $display("\t\ttime\tCK\tNull\tReset\tInput_R\t\tOut1");
        $monitor("%d\t%b\t%b\t%b\t%b\t\t%b",$time,CK,Null,Reset,Input_R,Out1);
    end

endmodule
```

# A.2. RTL Code and Test bench for the Query-Tree Protocol

## A.2.1. RTL Code for the Query-Tree Protocol

```
/*
Includes the sequential, combinational and ID memory of the Query-Tree Protocol for the last tag,
tag # 1111

Out1 is used to determine the status of the state:
Out1 = 1 ---> enquiry bit is same as the ID bit
Out1 = 0 ---> enquiry bit is different from ID bit, hence state is in S0
If tag # had been different from 1111, then additional port Out2 would have been needed such
that:
    Out1 = 1 & Out1 = 0 ---> present ID bit is 1
    Out1 = 0 & Out1 = 1 ---> present ID bit is 0
    Out1 = 0 & Out1 = 0 ---> state is in S0

Condition from S0 to S1 is when (Null && !Input_R) is true
Condition from S1 to S2 is when (Null && !Input_R) is true
*/

module query_last4 (CK,Null,Input_R,Reset, Out1);

parameter S0=2'b00;
parameter S1=2'b01;
parameter S2=2'b10;
input CK;      //System clock from the reader
input Null;          //Null from the reader
input Input_R;       //Received bit from the reader
input Reset;
output Out1;         //Out1 used to output the data
reg Out1;
//Out1 stores the current bit of the ID to compare it with the received bit from the reader
reg [3:0] ID_byte;
reg [1:0] State;
reg [1:0] index;
```

```
//Sequential part: Initialization and changing of states
always @ (posedge CK or posedge Reset)
begin
    if (Reset)
    begin
        State    <= S0;
        ID_byte  <= 4'b1111;
        index    <= 2'b11;
    end
    else
    begin
        case (State)
        S0:
            if (Null && !Input_R)
            begin
                if(index == 2'b11)
                begin
                    State    <= S1;
                    index    <= 2'b11;
                end
                else
                begin
                    State    <= S0;
                    index    <= 2'b11;
                end
            end
            else
            begin
                State    <= S0;
                index    <= index;
            end
        S1:
            if (Null && !Input_R)
            begin
                State    <= S2;
                index    <= index;
            end
            else if (Input_R ^ ID_byte[index])
            begin
                State    <= S0;
```

```
                index     <= index-1;
            end
            else
            begin
                State     <= S1;
                index     <= index-1;
            end
        S2:
            if (!index)
            begin
                State     <= S0;
                index     <=2'b11;
            end
            else
            begin
                State     <= S2;
                index     <= index-1;
            end
        default:
            begin
                State     <= S0;
                index     <= 2'b11;
            end
        endcase
    end
end

always @ (State or index)
begin
    if (State==S0)
    begin
        Out1 = 1'b0;
    end
    else
    begin
        Out1 = ID_byte[index];
    end
end

endmodule
```

## A.2.2. Test bench for the Query-Tree Protocol

```
//Tester for Tag # 1111 using the Query protocol

`timescale 1 ns / 1 ps

module query_last4_tb;

parameter in1=3, in2=4;
//in2 is used for the number of tags available, here n = 4
//in1 is used to limit the number of bits for commands, say For N = 4, n = 2:
//N-1-0-N, N-1-1-N; hence command needs 2 bit [in1-2:0], excluding N (Null) and
//N-1-0-0-N, N-1-0-1-N, N-1-1-0-N, and N-1-1-1-N; hence 3 bits [in1-1:0]

reg CK,Null,Input_R,Reset;
reg [in1-2:0] Input1;     //used for the 2-bit command
reg [in1-1:0] Input2;     //used for the 3-bit command
wire Out1;
integer index1, index2;

query_last4 u0(CK,Null,Input_R,Reset,Out1);

initial
begin
    CK = 0;
    Null = 0;
    Input_R = 0;
    Reset = 0;
    Input1 = 2'b10; //To simulate N-1-0-N and N-1-1-N commands
    Input2 = 3'b100; //To simulate N-1-0-0-N, N-1-0-1-N, N-1-1-0-N and N-1-1-1-N //commands
end

initial
begin
    $dumpfile ("query_last4_encounter.dump");
    $dumpvars (1,u0);
    repeat (1) @(negedge CK); Reset = 1;
    repeat (1) @(negedge CK); Reset = 0;     Null = 1; Input_R= 0; //To simulate N-N command
//Use 2 cycles before Null = 0, to implement the first Null-Null command
    repeat (2) @(negedge CK); Null = 0;
```

```
    repeat (4) @(negedge CK); Null = 1;        //4 cycles including S0 state
    repeat (1) @(negedge CK); Null = 0;

//The following for loop simulates the N-1-0-N and N-1-1-N commands
    for(index2=(in2/2)-1;index2>=0;index2=index2-1)
    begin
        Input_R = Input1[in1-2];
        for(index1=in1-3;index1>=0;index1=index1-1)
        begin
            Null = 0;
            repeat (1) @(negedge CK);
            Input_R = Input1[index1];
        end
        repeat (1) @(negedge CK);
        Input1 = Input1+1;
        Null = 1; Input_R= 0;              // to switch from S1 to S2
        repeat (1) @(negedge CK); Null = 0;
//after the rest of two clock cycles and S0
        repeat (2) @(negedge CK); Null = 1; Input_R= 0;
        repeat (1) @(negedge CK); Null = 0;
    end

//The following for loop simulates N-1-0-0-N, N-1-0-1-N, N-1-1-0-N, and N-1-1-1-N //commands
    begin: for1
        for(index2=in2-1;index2>=0;index2=index2-1)
        begin
            Input_R = Input2[in1-1];
            for(index1=in1-2;index1>=0;index1=index1-1)
            begin
                Null = 0;
                repeat (1) @(negedge CK);
                Input_R = Input2[index1];
            end
            repeat (1) @(negedge CK);
            Input2 = Input2+1;
            Null = 1; Input_R= 0;              // to switch from S1 to S2
            repeat (1) @(negedge CK); Null = 0;
            if (index2==0)      disable   for1;
            else
            begin
                repeat (1) @(negedge CK); Null = 1;   Input_R= 0;
```

```
    //after the rest of one clock cycle and S0
                repeat (1) @(negedge CK); Null = 0;
            end
        end
    end
    $dumpoff;
    Null = 0;
    repeat (1) @(negedge CK); Reset = 1;     //To show changing from S3 to S0
    #20 $finish;
end

always
    #25 CK = ~CK ;

initial
begin
    $display("\t\ttime\tCK\tNull\tReset\tInput_R\t\tOut1");
    $monitor("%d\t%b\t%b\t%b\t%b\t\t%b",$time,CK,Null,Reset,Input_R,Out1);
end

endmodule
```

# A.3. RTL Code and Test bench for the Improved Query-Tree Protocol

## A.3.1. RTL Code for the Improved Query-Tree Protocol

```
/*
Verilog Top-level module for Improved Query-Tree Protocol includes the sequential, combinational
and ID memory of the Improved Query-Tree Protocol for the last tag, tag # 1111

Out1 is used to determine the status of the state:
Out1 = 1 ---> enquiry bit is same as the ID bit
Out1 = 0 ---> enquiry bit is different from ID bit, hence state is in S0

Condition from S0 to S1 is when (Null && !Input_R) is true
Condition from S1 to S2 is when (Null && !Input_R) is true
Condition from S2 to S0 is when SS = (Null && Input_R) is true
*/

module impquery_last4 (CK,Null,Input_R,Reset, Out1);

parameter S0=2'b00;
parameter S1=2'b01;
parameter S2=2'b10;
input CK;       //System clock from the reader
input Null;           //Null from the reader
input Input_R;        //Received bit from the reader
input Reset;
output Out1;          //Out1 used to output the data
reg Out1;
//Out1 stores the current bit of the ID to compare it with the received bit from the reader
reg [3:0] ID_byte;
reg [1:0] State;
reg [1:0] index;

//Sequential part: Initialization and changing of states
always @ (posedge CK or posedge Reset)
begin
    if (Reset)
```

```
begin
    State    <= S0;
    ID_byte  <= 4'b1111;
    index    <= 2'b11;
end
else
begin
    case (State)
    S0:
        if (Null && !Input_R)
        begin
            if(index == 2'b11)
            begin
                State    <= S1;
                index    <= 2'b11;
            end
            else
            begin
                State    <= S0;
                index    <= 2'b11;
            end
        end
        else
        begin
            State    <= S0;
            index    <= index;
        end
    S1:
        if (Null && !Input_R)
        begin
            State    <= S2;
            index    <= index;
        end
        else if (Input_R ^ ID_byte[index])
        begin
            State    <= S0;
            index    <= index-1;
        end
        else
        begin
            State    <= S1;
```

```
                index     <= index-1;
            end
        S2:
            if (!index)
            begin
                State     <= S0;
                index     <= 2'b11;
            end
            else if (Null && Input_R)
            begin
                State     <= S0;
                index     <= 2'b11;
            end
            else
            begin
                State     <= S2;
                index     <= index-1;
            end
        default:
            begin
                State     <= S0;
                index     <= 2'b11;
            end
        endcase
    end
end

always @ (State or index)
begin
    if (State==S0)
    begin
        Out1 = 1'b0;
    end
    else
    begin
        Out1 = ID_byte[index];
    end
end

endmodule
```

## A.3.2. Test bench for Improved Query-Tree Protocol

```
//Tester for Tag # 1111 using the Improved Query protocol

`timescale 1 ns / 1 ps

module impquery_last4_tb;

parameter in1=3, in2=4;
reg CK,Null,Input_R,Reset;
reg [in1-2:0] Input1;
reg [in1-1:0] Input2;
wire Out1;
integer index1, index2;

impquery_last4 u0(CK,Null,Input_R,Reset,Out1);

initial
begin
    CK = 0;
    Null = 0;
    Input_R = 0;
    Reset = 0;
    Input1 = 2'b10;     Input2 = 3'b100;
end

initial
begin
    $dumpfile ("impquery_last4_encounter.dump");
    $dumpvars (1,u0);
    repeat (1) @(negedge CK); Reset = 1;
    repeat (1) @(negedge CK); Reset = 0;     Null = 1; Input_R = 0;
//Use 2 cycles before Null = 0, to implement the first Null-Null command
    repeat (2) @(negedge CK); Null = 0;
    repeat (2) @(negedge CK); Null = 1; Input_R = 1;   //Stop Send
    repeat (1) @(negedge CK); Input_R = 0;  //Prepare for S0 to S1
    repeat (1) @(negedge CK); Null = 0; //return to S0 then to S1

    for(index2=(in2/2)-1;index2>=0;index2=index2-1)
    begin
```

```
        Input_R = Input1[in1-2];
        for(index1=in1-3;index1>=0;index1=index1-1)
        begin
            Null = 0;
            repeat (1) @(negedge CK);
            Input_R = Input1[index1];
        end
        repeat (1) @(negedge CK);
        Input1 = Input1+1;
        Null = 1; Input_R = 0;          // to switch from S1 to S2
        repeat (1) @(negedge CK); Null = 0;
        repeat (1) @(negedge CK); Null = 1; Input_R = 1;//Stop Send, then from S2 to S0
        repeat (1) @(negedge CK);  Input_R = 0;   //Prepare for S0 to S1
        repeat (1) @(negedge CK);  Null = 0;  //return to S0 then to S1
    end

    begin: for1
        for(index2=in2-1;index2>=0;index2=index2-1)
        begin
            Input_R = Input2[in1-1];
            for(index1=in1-2;index1>=0;index1=index1-1)
            begin
                Null = 0;
                repeat (1) @(negedge CK);
                Input_R = Input2[index1];
            end
            repeat (1) @(negedge CK);
            Input2 = Input2+1;
            Null = 1; Input_R = 0;          // to switch from S1 to S2
            repeat (1) @(negedge CK); Null = 0;
        if (index2==0)      disable   for1;
        else
        begin
            repeat (1) @(negedge CK); Null = 1; //after the rest of one clock cycle and S0
            repeat (1) @(negedge CK); Null = 0;
        end
        end
    end

    $dumpoff;
    Null = 0;
```

```
    repeat (1) @(negedge CK); Reset = 1;     //To show changing from S3 to S0

    #20 $finish;
end

always
    #25 CK = ~CK ;

//To display inputs and output of the design in table form
initial
begin
    $display("\t\ttime\tCK\tNull\tReset\tInput_R\t\tOut1");
    $monitor("%d\t%b\t%b\t%b\t%b\t\t%b",$time,CK,Null,Reset,Input_R,Out1);
end

endmodule
```

# A.4. RTL Code and Test bench for the Query-Tree Protocol

## A.4.1. RTL Code for the Proposed Protocol

```
/*
Verilog Top-level module for the proposed Protocol includes the sequential, combinational and ID
memory of the Combined-Binary-Query-Tree Protocol for the last tag, tag # 1111

Out1 is used to determine the status of the state:
Out1 = 1 ---> enquiry bit is same as the ID bit
Out1 = 0 ---> enquiry bit is different from ID bit, hence state is in S0

1) Condition from S0 to S1 is when (Null && !Input_R) is true and when index == 2'b11
        Otherwise S0 to S0
2) Condition from S1 to S2 is when (Null && !Input_R) is true and when (Input_R ^ ID_byte[index])
is false:
        if Input_R ^ ID_byte[index] is true, S1 to S1
            Otherwise S1 to S0
3) Condition from S2 to S3 is when SS = (Null && Input_R) is true and when index != 0
        if SS == 0 and index != 0 then S2 to S2
            if index == 0, S2 to S0
4) Condition from S3 to S2 is when Input_R ^ ID_byte[index] is false
        Otherwise S3 to S0
*/


module combined_last4 (CK,Null,Input_R,Reset, Out1);

parameter S0=2'b00;
parameter S1=2'b01;
parameter S2=2'b10;
parameter S3=2'b11;
input CK;       //System clock from the reader
input Null;         //Null from the reader
input Input_R;      //Received bit from the reader
input Reset;
output Out1;        //Out1 used to output the data
```

```
reg Out1;
//Out1 stores the current bit of the ID to compare it with the received bit from the reader
reg [3:0] ID_byte;
reg [1:0] State;
reg [1:0] index;

//Sequential part: Initialization and changing of states
always @ (posedge CK or posedge Reset)
begin
    if (Reset)
    begin
        State    <= S0;
        ID_byte  <= 4'b1111;
        index    <= 2'b11;
    end
    else
    begin
        case (State)
        S0:
            if (Null && !Input_R)
            begin
                if(index == 2'b11)
                begin
                    State    <= S1;
                    index    <= 2'b11;
                end
                else
                begin
                    State    <= S0;
                    index    <= 2'b11;
                end
            end
            else
            begin
                State    <= S0;
                index    <= index;
            end
        S1:
            if (Null && !Input_R)
            begin
                State    <= S2;
```

```
            index     <= index;
        end
        else if (Input_R ^ ID_byte[index])
        begin
            State     <= S0;
            index     <= index-1;
        end
        else
        begin
            State     <= S1;
            index     <= index-1;
        end
    S2:
        if (!index)
        begin
            State     <= S0;
            index     <= 2'b11;
        end
        else if (Null && Input_R)
        begin
            State     <= S3;
            index     <= index;
        end
        else
        begin
            State     <= S2;
            index     <= index-1;
        end
    S3:
        if (Input_R ^ ID_byte[index])
        begin
            State     <= S0;
            index     <= 2'b11;
        end
        else
        begin
            State     <= S2;
            index     <= index-1;
        end
    default:
        begin
```

```
                State     <= S0;
                index     <= 2'b11;
            end
        endcase
    end
end

always @ (State or index)
begin
    if (State==S0)
    begin
        Out1 = 1'b0;
    end
    else
    begin
        Out1 = ID_byte[index];
    end
end

endmodule
```

## A.4.2. Test bench for the Proposed Protocol

```
//Tester for Tag # 1111 using the Combined-BQ Protocol

`timescale 1 ns / 1 ps

module combined_last4_tb;

parameter in1=3, in2=4;
reg CK,Null,Input_R,Reset;
reg [in1-2:0] Input1;
reg [in1-1:0] Input2;
wire Out1;
integer index1, index2;

combined_last4 u0(CK,Null,Input_R,Reset,Out1);

initial
    begin
        CK = 0;
        Null = 0;
        Input_R = 0;
        Reset = 0;
        Input1 = 2'b10;     Input2 = 3'b100;
    end

initial
    begin
        $dumpfile ("combined_last4_encounter.dump");
        $dumpvars (1,u0);
        repeat (1) @(negedge CK); Reset = 1;
        repeat (1) @(negedge CK); Reset = 0;     Null = 1; Input_R = 0;
//Use 2 cycles before Null = 0, to implement the first Null-Null command
        repeat (2) @(negedge CK); Null = 0;
        repeat (2) @(negedge CK); Null = 1; Input_R = 1;   //Stop Send
        repeat (1) @(negedge CK); Null = 0;Input_R = 0;
//send masking bit Input_R = 0, in S3.
        repeat (1) @(negedge CK); Null = 1; Input_R = 1;   //Stop Send
        repeat (1) @(negedge CK); Null = 0;Input_R = 0;    //send masking bit
        repeat (2) @(negedge CK); Null = 1; Input_R = 0;   //start 2nd round
```

```
        repeat (1) @(negedge CK); Null = 0;Input_R = 1;
        repeat (1) @(negedge CK); Null = 0;Input_R = 0;
        repeat (2) @(negedge CK); Null = 1;Input_R = 0;    //for S1 to S2 for tag # 10 or 11
        repeat (1) @(negedge CK); Null = 0; Input_R = 0;
//to let tag # 10 or 11 send their last bit
        repeat (1) @(negedge CK); Null = 1; Input_R = 0;   //start 3rd round
        repeat (1) @(negedge CK); Null = 0;Input_R = 1;
        repeat (2) @(negedge CK); Null = 1;Input_R = 0;    //received 11 hence jump to S2
        repeat (1) @(negedge CK); Null = 1; Input_R = 1;   //Stop Send
        repeat (1) @(negedge CK); Null = 0;Input_R = 0;    //send masking bit
        repeat (2) @(negedge CK); Null = 1; Input_R = 0;   //start 4th round
        repeat (1) @(negedge CK); Null = 0;Input_R = 1;
        repeat (3) @(negedge CK); Null = 1;Input_R = 0;    //received 111 hence jump to S2
        repeat (1) @(negedge CK); Null = 0;Input_R = 1;    //should go to S0 after this cycle

        $dumpoff;

        repeat (1) @(negedge CK); //To show changing from S2 to S0

        #20 $finish;
    end

always
    #25 CK = ~CK ;

//To display inputs and output of the design in table form
initial
    begin
        $display("\t\ttime\tCK\tNull\tReset\tInput_R\t\tOut1");
        $monitor("%d\t%b\t%b\t%b\t%b\t\t%b",$time,CK,Null,Reset,Input_R,Out1);
    end

endmodule
```

# Appendix B MATLAB Codes for performance comparison

Protocol level performance comparison of the four protocols was performed using MATLAB. Two MATLAB programs were written, the first of which compares Binary-tree, Query-tree, and Improved-Query-tree protocols and the second one compares Binary-tree, Improved-Query-tree, and Combined-Binary-Query-tree protocols. Figure 5.1 and Figure 5.2 were the result of the first program, while Figure 5.3 is the result of the second program. In both programs, the outputs are tables and graphs that show the values of values of number of transitions, clock cycles, number of transitions per clock cycles, and power for different values N.

# B.1. MATLAB Program for Comparison of Binary, Query, and Improved Query Protocols

```
% Mat lab code for comparison of Binary, Query, and Improved-Query
%Graph and Table for Binary, Query, and Improved-Query:
%Worst Case Analysis:
%Comparison of Binary, Query, and Improved-Query in terms of Number of Transitions, Clock
%Cycles, energy, and Power: from tag side, where the tag in question is the last tag to be read
%Variables used in this Matlab that do not correspond to the variables used in the analysis:
%    y is number of tags present (in the analysis "n" was used)
%    z is used to denote ceil(log2(y))-1 (in the analysis "l" was used)
%Equal clock cycles used to determine power
%Equivalent transitions are based on the Imp-Query Clock Cycle
%Table shows Transitions for Binary, Query, and Improved-Query and Graph shows Transitions,
%Clock Cycles, Energy, and Power for: Binary, Query, and Improved-Query
%modified on November 21, 2007

% 1) the first half of the program: Output will be in table form

clc;
Nmax = 7;
for N = 3:Nmax;                                    %N (number of bits)

    fprintf('\nNumber of ID bits = %2.0f\n',N)
    fprintf('Tags\tTb\t\tCkb\tTPCb\tTque\tCkq\t\tTPCq\tTiq\t\tCkiq\tTPCiq\tTeq_bin\tTeq_imq\tTeq
    _q\n')
```

```
fprintf('____\t__\t\t\t___\t____\t____\t___\t\t\t____\t___\t\t\t____\t_____\t_____\t_____\t_____\t___
__\n')

for y = 2:(2^N);          %The number of tags of the total tags
        if (y>=2^(N-1))
          x = log(2^(N-1))/log(2);
          x1 = (N+2)*(ceil(0.5*((2^(N-1))-3))+(2^(N-1))+1);
% x1 is part that adds in Timqclk
        else
          x = log(y)/log(2);
          x1 = (N+2)*(ceil(0.5*(y-3))+y+1);
        end
          z = ceil(x)-1;

        if ((y>=2)&&(y<=2^(N-2)))
            Tb = (2^(z+2))+2*y*(N-z-1)-1;
            Ckb = y*(2*N-1)+3;
        elseif ((y>2^(N-2)))&&(y<=2^(N-1))
            Tb = (2^(z+1))*(N-z)+(2*y-3);
            Ckb = y*(2*N-1);
        else
            Tb = (2^(z+1))*(N-z)+(2*(2^(N-1))-3);
            Ckb = (2^(N-1))*(2*N-1);
        end

       if (y>=2^(N-1))
       Tq = z+4*(2^(N-1));
```

```
        Ckq = (2*(2^(N-1))-1)*(N+2);
        Tiq = z+4*(2^(N-1));
            Ckiq = 0.0;
            for count2 = 0:(z-1);
              Ckiq = Ckiq +
((floor((2^(N-1)-(2^(count2+1))-1)/((2^(count2+2)))))+1)*(N+1-count2);
            end
            Ckiq = Ckiq + x1;
        else
      Tq = z+4*y;
      Ckq = (2*y-1)*(N+2);

      Tiq = z+4*y;
            Ckiq = 0.0;
            for count2 = 0:(z-1);
                Ckiq = Ckiq +
((floor((y-(2^(count2+1))-1)/((2^(count2+2)))))+1)*(N+1-count2);
            end
            Ckiq = Ckiq + x1;
        end

      TPCb = Tb/Ckb;
      TPCq = Tq/Ckq;
      TPCiq = Tiq/Ckiq;
      Teq_bin = Tb*(Ckb/Ckiq);
      Teq_q = Tq*(Ckq/Ckiq);
      Teq_imq = Tiq;
```

```
        table = [y;Tb;Ckb;TPCb;Tq;Ckq;TPCq;Tiq;Ckiq;TPCiq;Teq_bin;Teq_imq;Teq_q];

fprintf('%2.0f\t%5.0f\t%5.0f\t%2.3f\t%5.0f\t%5.0f\t%2.3f\t%5.0f\t%5.0f\t%2.3f\t%5.0f\t%5.0f\t%5.0f
\n',table)
    end
end

% 2) The 2nd half is used for plotting

Nmax = 10;   %Nmax = 15;
Nmin = 4;
Nmaxmin = Nmax-Nmin+1;
N = Nmin:Nmax;
value_ofy = 4; %value_ofy = 8; number of tags that exist simultaneously for reading can also be 8.
y = value_ofy;
for j=1:Nmaxmin;

    if (y>=2^(N(j)-1))
        y=2^(N(j)-1);
        x = log(y)/log(2);
        z = ceil(x)-1;
        x1 = (N+2)*(ceil(0.5*(y-3))+y+1);
    else
        x = log(y)/log(2);
        z = ceil(x)-1;
        x1 = (N+2)*(ceil(0.5*(y-3))+y+1);
    end
```

```
        if ((y>=2)&&(y<=2^(N(j)-2)))
            Tb(j) = (2^(z+2))+2*y*(N(j)-z-1)-1;
            Ckb(j) = y*(2*N(j)-1)+3;
        else
            Tb(j) = (2^(z+1))*(N(j)-z)+(2*y-3);
            Ckb(j) = y*(2*N(j)-1);
        end
    TPCb(j) = Tb(j)/Ckb(j);

    Tq(j) = z+4*y;
    Ckq(j) = (2*y-1)*(N(j)+2);
    TPCq(j) = Tq(j)/Ckq(j);

    Tiq(j) = z+4*y;
            Ckiq(j) = 0.0;
            for count2 = 0:(z-1);
          Ckiq(j) = Ckiq(j) +
((floor((y-(2^(count2+1))-1)/((2^(count2+2)))))+1)*(N(j)+1-count2);
            end
            Ckiq(j) = Ckiq(j) + x1(j);

    TPCiq(j) = Tiq(j)/Ckiq(j);
    Teq_bin(j) = Tb(j)*(Ckb(j)/Ckiq(j));
    Teq_q(j) = Tq(j)*(Ckq(j)/Ckiq(j));
    Teq_imq(j) = Tiq(j);
    y=value_ofy;
end
```

```
 figure
subplot(221)
plot (N,Tb)
hold on
plot (N,Tq, '*')
hold on
plot (N,Tiq, '+')
%title ('Binary, Query, and Improved-Query: No. of Transitions ')
xlabel ('Number of bits')
ylabel ('Number of Transitions')

subplot(222)
plot (N,Ckb)
hold on
plot (N,Ckq, '*')
hold on
plot (N,Ckiq, '+')
%title ('Binary, Query, and Improved-Query: Clock Cycles')
xlabel ('Number of bits')
ylabel ('Number of Clock cycles')

subplot(223)
plot (N,TPCb,'-')
hold on
plot (N,TPCq, '*')
hold on
plot (N,TPCiq, '+')
```

```
%title ('Binary, Query, and Improved-Query: Transition per Clock Cylces')
xlabel ('Number of bits')
ylabel ('Transitions per Clock cycle')

subplot(224)
plot (N,Teq_bin,'-')
hold on
plot (N,Teq_q, '*')
hold on
plot (N,Teq_imq, '+')
%title ('Power with equal Clock Cylces')
xlabel ('Number of bits')
ylabel ('Power')
```

## B.2. MATLAB Program for Comparison of Binary, Improved-Query, and Combined-Binary Protocols

```
% Mat lab code for comparison of Binary, Improved-Query, and Improved-Binary-
%Query (or the proposed protocol)
%Graph and Table for Binary, Improved-Query, and Improved-Binary-Query:
%Worst Case Analysis: Power Calculation
%Comparison of Binary, Improved-Query, and Improved-Binary-Query in terms of Number of
%Transitions, Clock Cycles, energy, and Power: from tag side, where the tag in question is the last
%tag to be read
%y is number of tags present and z is used to denote ceil(log2(y))-1
%Equal clock cycles used to determine power
%Equivalent transitions are based on the Imp-Query Clock Cycle
%Table shows Transitions for Binary, Improved-Query, and Improved-Binary-Query.
%Graph shows Transitions, Clock Cycles, Energy, and Power for:
%Binary, Improved-Query, and Improved-Binary-Query protocols
%Modified on November 22, 2007

% 1) the first half of the program: Output will be in table form

clc;
Nmax = 7;
for N = 3:Nmax;                                    %N (number of bits)

    fprintf('\nNumber of ID bits = %2.0f\n',N)
    fprintf('Tags\tTb\t\tCkb\tTPCb\tTcbq\tCkcbq\t\tTPCcbq\tTiq\t\tCkiq\tTPCiq\tTeq_bin\tTeq_imq
    \tTeq_cbq\n')
```

```
fprintf('____\t__\t\t___\t____\t____\t___\t\t____\t___\t\t____\t_____\t_______\t_______\t___
__\n')

for y = 2:(2^N);          %The number of tags of the total tags

    if (y>=2^(N-1))
      x = log(2^(N-1))/log(2);
      x1 = (N+2)*(ceil(0.5*((2^(N-1))-3))+(2^(N-1))+1);
    else
      x = log(y)/log(2);
      x1 = (N+2)*(ceil(0.5*(y-3))+y+1);
    end
      z = ceil(x)-1;

    if ((y>=2)&&(y<=2^(N-2)))
       Tb = (2^(z+2))+2*y*(N-z-1)-1;
       Ckb = y*(2*N-1)+3;
    elseif ((y>2^(N-2)))&&(y<=2^(N-1))
       Tb = (2^(z+1))*(N-z)+(2*y-3);
       Ckb = y*(2*N-1);
    else
       Tb = (2^(z+1))*(N-z)+(2*(2^(N-1))-3);
       Ckb = (2^(N-1))*(2*N-1);
    end

   if (y>=2^(N-1))
          %Improved-Binary-Query Section or combined-Binary-Query
```

```
        Tibq = (2^(z+1)) + (2-(2^(N-1)))*z + N*(2^(N-1))+2;
        Tibq = Tibq - ((2^(z+1))+(2^(N-1))*(N-z-2)-1);
        %Where, T11+T12 = ((2.^(z+1))+y*(N-z-2)-1)

        Ckibq = (2^(N-1))*(N+3)-1;
        Tiq = z+4*(2^(N-1));
        %Where, T11+T12 = ((2.^(z+2))+2*y*(N-z-2)-N+z);
            Ckiq = 0.0;
            for count2 = 0:(z-1);
              Ckiq = Ckiq +
((floor((2^(N-1)-(2^(count2+1))-1)/((2^(count2+2)))))+1)*(N+1-count2);
            end
            Ckiq = Ckiq + x1;
         else
    %Improved-Binary-Query Section
        Tibq = (2^(z+1)) + (2-y)*z + N*y +2;
        Tibq = Tibq - ((2^(z+1))+y*(N-z-2)-1);

        Ckibq = y*(N+3)-1;
        Tiq = z+4*y;
                Ckiq = 0.0;
                for count2 = 0:(z-1);
                    Ckiq = Ckiq +
((floor((y-(2^(count2+1))-1)/((2^(count2+2)))))+1)*(N+1-count2);
                end
                Ckiq = Ckiq + x1;
         end
```

```
        TPCb = Tb/Ckb;
        TPCibq = Tibq/Ckibq;
        TPCiq = Tiq/Ckiq;
        Teq_bin = Tb*(Ckb/Ckiq);
        Teq_ibq = Tibq*(Ckibq/Ckiq);
        Teq_imq = Tiq;

        table = [y;Tb;Ckb;TPCb;Tibq;Ckibq;TPCibq;Tiq;Ckiq;TPCiq;Teq_bin;Teq_imq;Teq_ibq];
fprintf('%2.0f\t%5.0f\t%5.0f\t%2.3f\t%5.0f\t%5.0f\t%2.3f\t%5.0f\t%5.0f\t%2.3f\t%5.0f\t%5.0f\t%5.0f
\n',table)
    end
end

% 2) the 2nd half of the program: Output will be in graph form

Nmax = 10;   %Nmax = 15;
Nmin = 4;
Nmaxmin = Nmax-Nmin+1;
N = Nmin:Nmax;
value_ofy = 4;      %value_of y = 8;
y = value_ofy;

for j=1:Nmaxmin;
    if (y>=2^(N(j)-1))
        y=2^(N(j)-1);
        x = log(y)/log(2);
        z = ceil(x)-1;
```

```
        x1 = (N+2)*(ceil(0.5*(y-3))+y+1); %part that adds in Tiq
    else
        x = log(y)/log(2);
        z = ceil(x)-1;
        x1 = (N+2)*(ceil(0.5*(y-3))+y+1); %part that adds in Tiq
    end
        if ((y>=2)&&(y<=2^(N(j)-2)))
            Tb(j) = (2^(z+2))+2*y*(N(j)-z-1)-1;
            Ckb(j) = y*(2*N(j)-1)+3;
        else
            Tb(j) = (2^(z+1))*(N(j)-z)+(2*y-3);
            Ckb(j) = y*(2*N(j)-1);
        end
        TPCb(j) = Tb(j)/Ckb(j);
%Improved-Binary-Query Section
        Tibq(j) = (2^(z+1)) + (2-y)*z + N(j)*y +2;
        Tibq(j) = Tibq(j) - ((2^(z+1))+y*(N(j)-z-2)-1);

        Ckibq(j) = y*(N(j)+3)-1;
        TPCibq(j) = Tibq(j)/Ckibq(j);

        Tiq(j) = z+4*y;
                Ckiq(j) = 0.0;
                for count2 = 0:(z-1);
              Ckiq(j) = Ckiq(j) +
((floor((y-(2^(count2+1))-1)/((2^(count2+2)))))+1)*(N(j)+1-count2);
                end
```

```
                Ckiq(j) = Ckiq(j) + x1(j);


        TPCiq(j) = Tiq(j)/Ckiq(j);
        Teq_bin(j) = Tb(j)*(Ckb(j)/Ckiq(j));
        Teq_ibq(j) = Tibq(j)*(Ckibq(j)/Ckiq(j));
        Teq_imq(j) = Tiq(j);
        y=value_ofy;
end


figure
subplot(221)
plot (N,Tb)
hold on
plot (N,Tibq, '*')
hold on
plot (N,Tiq, '+')
%title ('Binary, Improved-Query, and Improved-Binary-Query: No. of Transitions ')
xlabel ('Number of bits(N)')
ylabel ('Number of State Transitions')


subplot(222)
plot (N,Ckb)
hold on
plot (N,Ckibq, '*')
hold on
plot (N,Ckiq, '+')
%title ('Binary, Improved-Query, and Improved-Binary-Query: Clock Cylces')
```

```
xlabel ('Number of bits(N)')
ylabel ('Number of Clock Cycles')

subplot(223)
plot (N,TPCb,'-')
hold on
plot (N,TPCibq, '*')
hold on
plot (N,TPCiq, '+')
%title ('Binary, Improved-Query, and Improved-Binary-Query: Transition per Clock Cylces')
xlabel ('Number of bits(N)')
ylabel ('Transitions per Clock Cycle')

subplot(224)
plot (N,Teq_bin,'-')
hold on
plot (N,Teq_ibq, '*')
hold on
plot (N,Teq_imq, '+')
%title ('Binary, Improved-Query, and Improved-Binary-Query: Power with equal Clock Cylces')
xlabel ('Number of bits(N)')
ylabel ('Power')
```

# Appendix C Permission from Co-authors and IEEE

In this section a letter that permits the author of this thesis to use papers that were co-written by Dr. C. Chen and Dr. Q. M. J. Wu is attached.

Also attached is an E-mail correspondence that permits the author to reuse IEEE-Copyrighted material.

## Permission to Use Previously Submitted/Accepted Papers

I Dr. Chunhong Chen and I Dr. Q. M. Jonathan Wu give permission to Mr. Mohammed Berhea to include the following papers into his master's thesis.

1) Paper accepted for poster presentation in ISCAS'08, May 2008, Seattle, USA, entitled:

M. Berhea, C. Chen, and Q. M. J. Wu, "Protocol-Level Performance Analysis for Anti-Collision Protocols in RFID Systems", In Proceedings of 2008 IEEE International Symposium on Circuits and Systems (ISCAS'08), May 2008, Seattle, USA

2) Paper submitted for publication to IEEE Trans. On Vehicular Technology, entitled:

M. Berhea, C. Chen, and Q. M. J. Wu, "Protocol-Level Performance Analysis and Implementation for Anti-Collision Protocols in RFID Systems"

Sincerely,

Dr. Chunhong Chen June 24, 2008

Dr. Q. M. Jonathan Wu

**From:** j.hansson@ieee.org
**Subject:** Re: Permission to Reuse IEEE-Copyrighted Material
**Date:** Tue, 24 Jun 2008 13:53:55 -0400
**To:** "Berhea M" <berhea@uwindsor.ca>

Comments/Response to Case ID: 00625C1C

ReplyTo: Copyrights@ieee.org

From: Jacqueline Hansson Date: 06/24/2008

Subject: Re: Permission to Reuse IEEE-Copyrighted Material

Send To: "Berhea M" <berhea@uwindsor.ca>

cc:

Dear Mohammed Berhea:

This is in response to your letter below, in which you have requested permission to reprint, in your upcoming thesis/dissertation, your described IEEE copyrighted material, We are happy to grant this permission.

In regards to distributing paper copies of your thesis or placing it on the university's website, please be sure that the following copyright/credit notice appears in the credits with the appropriate details filled in:

© 2008 IEEE. Reprinted, with permission, from (complete publication information).

Additionally, in regards to placing your thesis on the university's website, at the beginning of the credits or in an appropriate and prominent place on the website the following message should be displayed:

This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of the University of Windsor's products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by writing to pubs-permissions@ieee.org.

By choosing to view this material, you agree to all provisions of the copyright laws protecting it.

Please be advised that wherever a copyright notice from another organization is displayed beneath a figure, a photo, a videotape or a Powerpoint presentation, you must get permission from that organization, as IEEE would not be the copyright holder.

Sincerely,

Jacqueline Hansson

© © © © © © © © © © © © © © © © © © ©

IEEE Intellectual Property Rights Office
445 Hoes Lane
Piscataway, NJ 08855-1331 USA
+1 732 562 3966 (phone)
+1 732 562 1746 (fax)

IEEE-- Fostering technological innovation
and excellence for the benefit of humanity.
© © © © © © © © © © © © © © © © ©

Dear Sir/Madam:

I am requesting a written permission for the following paper to include it into my Master's Thesis.

The paper is written by me under the supervision of my professor, Dr. Chunhong Chen and was accepted for presentation at the 2008 IEEE International Symposium on Circuits and Systems, held in Seattle, USA, from May 18-21, 2008.

The information of the paper is described below:

1) Author's name: Mohammed Berhea, Chunhong Chen, and Q. M. Jonathan Wu.

2) Paper title: Protocol-Level Performance Analysis for Anti-Collision Protocols in RFID Systems

3) IEEE publication title: M. Berhea, C. Chen, Q. M. J. Wu "Protocol-Level Performance Analysis for Anti-Collision Protocols in RFID Systems", In Proceedings of 2008 IEEE International Symposium on Circuits and Systems (ISCAS'08), May 2008, Seattle, USA, pp.

I would like to reuse all portions of the paper into my thesis.

Thank you,

Sincerely,

Mohammed Berhea

Master's Candidate

Universityof Windsor
401 Sunset Avenue, Windsor, Ontario
CANADAN9B 3P4
Tel: (519) 253-3000 ext. 2542
Email: berhea@uwindsor.ca

# VITA AUCTORIS

Mohammed Berhea is graduated from the University of Windsor, where he obtained his B.Sc in Electrical Engineering. He is currently a candidate for the Master's degree in Electrical and Computer Engineering department at the University of Windsor and hopes to graduate in August 2008.